4D v11 SQL

Langage
Windows® / Mac OS®

4D v11 SQL - Langage
Release 4 (11.4) pour Windows® et Mac OS®

Les informations contenues dans ce manuel peuvent faire l'objet de modifications sans préavis et ne sauraient en aucune manière engager 4D SAS. La fourniture du logiciel décrit dans ce manuel est régie par un octroi de licence dont les termes sont précisés par ailleurs dans la licence électronique figurant sur le support du Logiciel et de la Documentation y afférente. Le logiciel et sa Documentation ne peuvent être utilisés, copiés ou reproduits sur quelque support que ce soit et de quelque manière que ce soit, que conformément aux termes de cette licence.

Aucune partie de ce manuel ne peut être reproduite ou recopiée de quelque manière que ce soit, électronique ou mécanique, y compris par photocopie, enregistrement, archivage ou tout autre procédé de stockage, de traitement et de récupération d'informations, pour d'autres buts que l'usage personnel de l'acheteur, et ce exclusivement aux conditions contractuelles, sans la permission explicite de 4D SAS.

Sommaire

10. Débogueur....409

11. Définition structure....453

12. Documents système..... 477

13. Enregistrements..... 539

14. Enregistrements (verrouillage).. 563

15. Ensembles.....581

16. Environnement 4D..... 611

18. Etats rapides..... 711

29. Images..... 1011

30. Import-Export.....1049

31. Impressions.....1067

32. Interface utilisateur..... 1121

34. Langage..... 1191

35. Liens.....1213

36. List Box..... 1239

37. Listes hiérarchiques..... 1277

46. Propriétés des objets.....1555

47. Protocole sécurisé..... 1605

48. Recherches et tris..... 1613

49. Ressources..... 1655

52. Sélections.....1739

53. Sélections Temporaires..... 1775

54. Serveur Web..... 1787

55. Sous-enregistrements.....1951

56. SQL..... 1971

68. Zone Web..... 2293

1

Introduction

4D possède son propre langage de programmation. Ce langage intégré, qui comprend plus de 1000 commandes, fait de 4D un outil de développement très puissant.

Le langage de 4D peut être utilisé pour de multiples types de tâches, de la réalisation de calculs simples à la création d'interfaces utilisateur personnalisées complexes. Par l'intermédiaire des routines du langage, vous pourrez par exemple :

• Accéder par programmation à tous les éditeurs de gestion des enregistrements (tris, recherches...),
• Créer et imprimer des états et des étiquettes complexes avec les données de la base,
• Communiquer avec d'autres systèmes d'information,
• Gérer des documents,
• Importer et exporter des données entre des bases 4D et d'autres applications,
• Incorporer des procédures écrites dans d'autres langages que celui de 4D.

La flexibilité et la puissance du langage de 4D en font l'outil idéal pour toute une gamme de fonctions de gestion de l'information. Les utilisateurs novices peuvent rapidement effectuer des calculs. Les utilisateurs expérimentés peuvent personnaliser leurs bases sans devoir connaître la programmation. Les développeurs plus expérimentés peuvent utiliser la puissance du langage de 4D pour ajouter à leurs bases de données des fonctions sophistiquées, allant jusqu'au transfert de fichiers et aux communications. Les développeurs maîtrisant la programmation dans d'autres langages peuvent ajouter leurs propres routines au langage de 4D.

Le langage de programmation de 4D s'enrichit lorsqu'un plug-in est installé dans l'application. Chaque plug-in comporte en effet des commandes spécifiques, permettant de gérer les fonctions qu'il accomplit.

A propos des manuels

Les manuels de 4D, listés ci-dessous, s'appliquent indifféremment à décrire le fonctionnement de 4D et de 4D Server. Une seule exception : le "Manuel de référence 4D Server", qui traite uniquement des fonctions de 4D Server.

• Le manuel "Langage" est le guide de référence du langage de 4D. Il vous apprend à personnaliser votre base en utilisant les commandes et fonctions de 4D.
• Le manuel "Mode Développement" détaille les éditeurs et les outils disponibles dans l'environnement de développement.

• Le manuel "Autoformation" vous propose de suivre au pas à pas des exemples de création et d'utilisation de bases de données. Ces exercices vous permettent de vous familiariser rapidement avec les fonctionnalités et les concepts de 4D et 4D Server.
• Le "Manuel de référence 4D Server" est consacré à l'installation et la gestion de bases de données multi-utilisateurs avec 4D Server.

A propos de ce manuel

Ce manuel décrit le langage de 4D. Nous supposons que vous êtes familiarisé avec le vocabulaire élémentaire utilisé dans 4D tels que tables, champs ou formulaires. Avant de lire ce manuel, nous vous conseillons de :

• découvrir 4D à travers les exemples du manuel "Autoformation",
• commencer à créer vos propres bases de données, en vous référant au manuel "Mode Développement",
• élargir vos connaissances en étudiant les nombreuses bases de démonstration et d'exemple disponibles notamment sur le site Web de 4D (http://www.4d.fr).

Conventions d'écriture

Dans ce manuel, diverses conventions d'écriture sont employées :

• à l'instar de l'éditeur de méthodes de 4D, les commandes sont écrites en majuscules et en caractères spéciaux : OUVRIR DOCUMENT. Les fonctions (commandes renvoyant une valeur) ont une lettre initiale en majuscule et sont écrites en minuscule : Remplacer caracteres.
• Dans la syntaxe des commandes, les caractères { } (accolades) indiquent des paramètres facultatifs. Par exemple, SIECLE PAR DEFAUT (siècle{; anPivot}) signifie que le paramètre anPivot peut être omis lors de l'appel de la commande.
• Dans la syntaxe des commandes, le caractère | indique une alternative. Par exemple, Table (numTable | unPtr) indique que cette fonction accepte soit un numéro de table soit un pointeur comme paramètre.
• Dans certains exemples de cette documentation, une ligne de code peut se prolonger sur une plusieurs lignes, par manque de place. Toutefois, tapez ces exemples sans appuyer sur la touche Retour chariot, en une seule ligne de code.

Pour en savoir plus...

Si vous lisez ce manuel pour la première fois, reportez-vous à la section Introduction.

Cette section présente le langage de 4D. Elle apporte des réponses aux questions suivantes :

- Qu'est-ce que le langage de 4D, et que peut-il vous apporter ?
- Comment utiliser les méthodes ?
- Comment développer une application avec 4D ?

Ces sujets sont évoqués d'un point de vue général ; chacun d'entre eux est traité plus en détail dans les autres sections de ce manuel.

Qu'est-ce qu'un langage ?

Le langage de 4D n'est pas très différent de celui que nous utilisons tous les jours. C'est une forme de communication servant à exprimer des idées, informer ou donner des instructions. Tout comme un langage parlé, celui de 4D se compose d'un vocabulaire, d'une grammaire et d'une syntaxe, que vous employez pour indiquer au programme comment gérer votre base et vos données.

Il n'est pas nécessaire de connaître entièrement le langage. Pour pouvoir vous exprimer en anglais, vous n'êtes pas obligé de connaître la totalité de la langue anglaise ; en réalité, un peu de vocabulaire suffit. Il en va de même pour 4D — il vous suffit de connaître un minimum du langage pour être productif, vous en apprendrez de plus en plus au fur et à mesure de vos besoins.

Pourquoi utiliser un langage ?

A première vue, un langage de programmation dans 4D peut sembler inutile. En effet, 4D propose en mode Développement des outils puissants et entièrement paramétrables, permettant d'effectuer une grande variété de tâches de gestion des données. Les opérations fondamentales telles que la saisie de données, les recherches, les tris et la création d'états sont effectuées facilement. De nombreuses autres possibilités sont aussi disponibles, telles que les filtres de validation des données, les aides à la saisie, la représentation graphique et la génération d'étiquettes.

Alors, pourquoi avons-nous besoin d'un langage ? Voici quelques-uns de ses principaux rôles :

• Automatiser les tâches répétitives : par exemple la modification de données, la génération d'états complexes et la réalisation de longues séries d'opérations ne nécessitant pas l'intervention d'un utilisateur.
• Contrôler l'interface utilisateur : vous pouvez gérer les fenêtres, les menus, contrôler les formulaires et les objets d'interface.
• Gérer les données de manière très fine : cette gestion inclut le traitement de transactions, les systèmes de validation complexes, la gestion multi-utilisateurs, l'utilisation des ensembles et des sélections temporaires.
• Contrôler l'ordinateur : vous pouvez contrôler les communications par le port série, la gestion des documents et des erreurs.
• Créer des applications : vous pouvez créer des bases de données personnalisées, faciles à utiliser, en mode Application.
• Ajouter des fonctionnalités au serveur Web 4D intégré : par exemple, vous pouvez créer des pages HTML dynamiques et les insérer parmi celles qui sont automatiquement créées par 4D lorsque les formulaires sont convertis.

Le langage vous permet de contrôler totalement la structure et le fonctionnement de votre base de données. 4D propose de puissants éditeurs "génériques", mais le langage vous offre la possibilité de personnaliser autant que vous voulez votre base de données.

Prendre le contrôle de vos données

Le langage de 4D vous permet de prendre le contrôle total de vos données, d'une manière à la fois puissante et élégante. Il reste d'un abord facile pour un débutant, et est suffisamment riche pour un développeur d'applications. Il permet de passer par étapes d'une simple exploitation des fonctions intégrées de 4D à une base entièrement personnalisée.

Les commandes du langage de 4D vous laissent la possibilité d'accéder aux éditeurs standard de gestion des enregistrements. Par exemple, lorsque vous utilisez la commande CHERCHER, vous accédez à l'éditeur de recherches (accessible en mode Développement via la commande **Chercher...** dans le menu **Enregistrements**. Vous pouvez, si vous le voulez, indiquer explicitement à la commande CHERCHER ce qu'elle doit rechercher. Par exemple, CHERCHER([Personnes]Nom="Dupont") trouvera toutes les personnes nommées Dupont dans votre base.

Le langage de 4D est très puissant — une commande remplace souvent des centaines, voire des milliers de lignes de code écrites dans les langages informatiques traditionnels. Et cette puissance s'accompagne d'une réelle simplicité: les commandes sont écrites en français. Par exemple, vous utilisez la commande CHERCHER pour effectuer une recherche ; pour ajouter un nouvel enregistrement, vous appelez la commande AJOUTER ENREGISTREMENT.

Le langage est organisé de telle manière que vous puissiez facilement accomplir n'importe quelle tâche. L'ajout d'un enregistrement, le tri d'enregistrements, la recherche de valeurs et les autres opérations de même type sont définies par des commandes simples et directes. Mais les routines peuvent également contrôler les ports série, lire des documents sur le disque, traiter des systèmes de transactions complexes, et plus encore.

Même les opérations les plus compliquées se définissent de manière relativement simple. Il serait inimaginable de les réaliser sans le langage de 4D. Cependant, même à l'aide de la puissance des commandes du langage, certaines tâches peuvent se révéler complexes et difficiles. Ce n'est pas l'outil lui-même qui fait le travail ; une tâche peut représenter en soi une difficulté, l'outil ne fait que faciliter son traitement. Par exemple, un logiciel de traitement de texte permet d'écrire un livre plus rapidement et plus facilement, mais il ne l'écrira pas à votre place. Le langage de 4D vous permet de gérer plus facilement vos données et d'appréhender les tâches ardues en toute confiance.

Est-ce un langage informatique "traditionnel" ?

Si vous êtes familier avec les langages informatiques traditionnels, ce paragraphe peut vous intéresser. Si ce n'est pas le cas, vous pouvez passer directement au paragraphe suivant.

Le langage de 4D n'est pas un langage informatique traditionnel. C'est un des langages les plus souples et les plus innovants que l'on puisse trouver aujourd'hui sur micro-ordinateur. Le langage a été conçu pour s'adapter à vous, et non l'inverse.

Pour utiliser les langages traditionnels, vous devez avant tout réaliser des maquettes détaillées. C'est même l'une des principales phases d'un développement. 4D vous permet de commencer à utiliser le langage à tout moment et dans n'importe quelle partie de votre base. Vous pouvez commencer par ajouter une méthode objet dans un formulaire, puis plus tard une ou deux méthodes formulaires. Comme votre base devient plus sophistiquée, vous pouvez ajouter une méthode projet contrôlée par un menu. Vous êtes totalement libre d'utiliser une petite partie du langage ou une partie plus étendue. Ce n'est pas "tout ou rien", comme c'est le cas dans beaucoup d'autres bases de données.

Les langages traditionnels vous obligent à définir et pré-déclarer vos objets sous une forme syntaxique rigide. Dans 4D, vous créez simplement un objet, comme un bouton, et vous l'utilisez. 4D gère automatiquement l'objet pour vous. Par exemple, pour utiliser un bouton, il suffit de le dessiner dans un formulaire et de lui donner un nom. Lorsque l'utilisateur clique sur le bouton, le langage déclenche automatiquement vos méthodes.

Les langages traditionnels sont souvent rigides et inflexibles, et exigent que les commandes soient saisies dans un style très formel et contraignant. Le langage de 4D rompt avec la tradition, au grand bénéfice de l'utilisateur.

Les méthodes, portes d'entrée du langage

Une méthode est une série d'instructions qui provoquent l'accomplissement d'une tâche par 4D. Toute méthode contient une ou plusieurs lignes d'instructions. Chaque ligne d'instruction est composée d'éléments du langage.

Puisque vous avez lu le manuel "Autoformation" de 4D (vous l'avez lu, n'est-ce pas ?), vous avez déjà écrit et utilisé des méthodes objet et des méthodes projet.

Dans 4D, vous pouvez créer cinq types de méthodes : des méthodes objet, des méthodes formulaire, des méthodes table ou "triggers", des méthodes projet et des méthodes base.

- **Méthode objet :** méthode généralement courte utilisée pour contrôler un objet de formulaire.
- **Méthode formulaire :** méthode qui gère l'affichage ou l'impression d'un formulaire.
- **Méthode table/trigger :** méthode utilisée pour appliquer les règles de fonctionnement de votre base.
- **Méthode projet :** méthode s'appliquant à la totalité de la base. Les méthodes projet peuvent être associées à des commandes de menus, par exemple.
- **Méthode base :** méthode utilisée pour initialiser des valeurs ou effectuer des actions particulières à l'ouverture ou à la fermeture d'une base, ou lorsqu'un navigateur Web se connecte à une base publiée comme serveur Web sur un réseau Intranet ou sur Internet.

Les paragraphes suivants présentent chaque type de méthode et expliquent brièvement la manière dont vous pouvez les utiliser pour automatiser votre base.

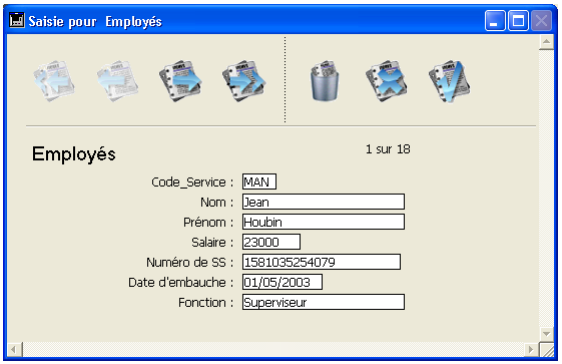
Démarrer avec les méthodes objet

Tout objet d'un formulaire pouvant déclencher une action — c'est-à-dire tout objet actif — peut être associé à une méthode objet. Une méthode objet surveille et gère l'objet actif lors de la saisie et de l'impression des données. Une méthode objet reste liée à un objet actif même lorsque l'objet est copié et collé. Ce principe vous permet de créer des librairies d'objets actifs réutilisables. La méthode objet s'exécute lorsque c'est nécessaire.

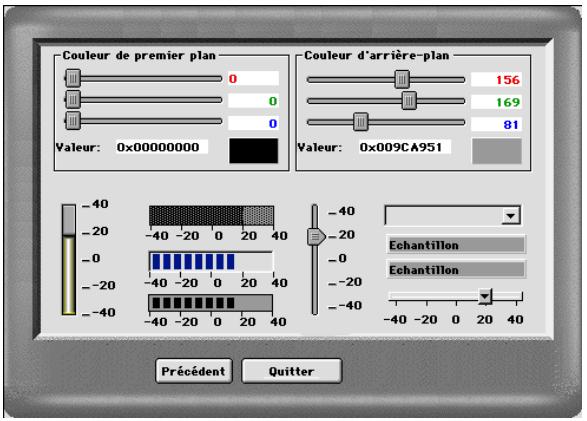
Les méthodes objet sont les outils de base pour la gestion de l'interface utilisateur. L'interface utilisateur est l'ensemble des moyens et des conventions par lesquels l'ordinateur communique avec l'utilisateur. L'interface utilisateur est la porte par laquelle vous entrez dans votre base de données. L'objectif est de la rendre aussi simple d'emploi que possible. L'interface utilisateur doit faire en sorte que l'interaction avec l'ordinateur soit agréable, que l'utilisateur l'apprécie ou, mieux, ne la remarque même pas.

Il y a deux types principaux d'objets actifs dans un formulaire : les objets actifs de saisie, d'affichage et de stockage des données, tels que les champs et les sous-champs, et les objets actifs de contrôle comme les zones de saisie, les boutons, les listes déroulantes et les thermomètres.

4D vous permet de construire des formulaires sobres et classiques, tel que celui présenté ci-dessous :



4D vous permet également de construire des formulaires comportant de multiples éléments de contrôle, comme celui-ci :



Vous pouvez aussi créer des formulaires originaux en laissant libre cours à votre imagination et en utilisant des éléments graphiques exubérants.

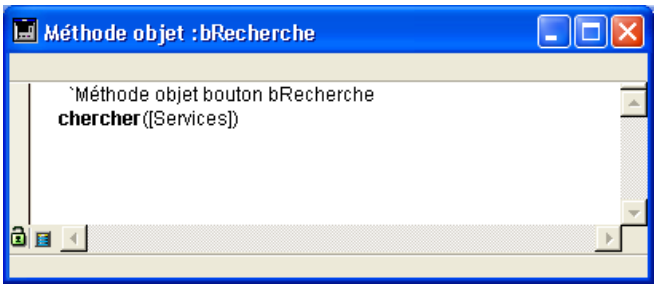
Vous n'êtes limité que par votre imagination ou par les souhaits des commanditaires de l'application :

Lorsque vous construisez des formulaires, rappelez-vous que tous les objets actifs disposent de systèmes de contrôle intégrés, comme les intervalles de valeurs et les filtres pour les zones de saisie, ou les actions automatiques pour les objets, menus et boutons. Essayez toujours d'utiliser ces systèmes intégrés avant de recourir aux méthodes objet. Ils sont comparables aux méthodes objet dans la mesure où ils restent associés aux objets actifs et ne sont exécutés que lorsque l'objet est sollicité. En général, vous utiliserez une combinaison d'aides intégrées et de méthodes objet pour contrôler l'interface utilisateur.

Typiquement, une méthode associée à un objet actif de saisie aura un rôle de gestion des données spécifiquement lié au champ ou à la variable. La méthode peut effectuer un contrôle ou un formatage des valeurs, ou des calculs. Elle peut également récupérer des informations en provenance d'autres tables. Bien entendu, certaines de ces tâches peuvent aussi être exécutées par l'intermédiaire des systèmes intégrés de contrôle de saisie des objets. N'utilisez les méthodes objet que lorsque l'opération à effectuer est plus complexe que ce que permettent ces systèmes. Pour plus d'informations sur les systèmes intégrés de contrôle de saisie, reportez-vous au manuel "*Mode Développement*" de 4D.

Des méthodes objet peuvent aussi être associées aux objets actifs de contrôle tels que les boutons. Ces objets sont essentiels pour la navigation au sein de votre base. Les boutons vous permettent de vous déplacer d'un enregistrement à l'autre, de changer de formulaire, d'ajouter et d'effacer des données. Ces objets actifs simplifient l'utilisation d'une base et réduisent le temps d'apprentissage. Les boutons disposent également de systèmes d'aides intégrés et, comme pour la saisie de données, il est préférable de les utiliser avant d'écrire des méthodes objet. Les méthodes vous permettent d'associer à vos objets des actions qui n'existent pas en standard.

Dans l'exemple ci-dessous, la méthode objet du bouton affichera l'éditeur de recherches lorsque l'utilisateur cliquera dessus.



A mesure que vous vous familiariserez avec les méthodes objet, vous pourrez créer des librairies d'objets avec leurs méthodes associées. Vous pourrez copier et coller ces objets entre différents formulaires, tables et bases. Vous pourrez même les conserver dans l'Album, prêts à être utilisés suivant vos besoins.

Contrôler des formulaires à l'aide des méthodes formulaire

Tout comme une méthode objet est associée à un objet actif d'un formulaire, une méthode formulaire est associée à un formulaire. Chaque formulaire peut comporter au plus une méthode formulaire.

Un formulaire est un modèle dans lequel vous pouvez visualiser vos données. Les formulaires vous permettent de présenter les données à l'utilisateur de différentes manières. Grâce aux formulaires, vous pouvez créer des écrans de saisie et des états récapitulatifs attractifs et faciles à utiliser. Une méthode formulaire contrôle et gère l'utilisation d'un formulaire spécifique, à la fois pour la saisie et l'impression des données.

Une méthode formulaire gère un formulaire à un plus haut niveau que les méthodes objet. Une méthode objet n'est exécutée que lorsque l'objet est utilisé, alors qu'une méthode formulaire est exécutée lorsque n'importe quel objet du formulaire est activé. Les méthodes formulaire sont généralement utilisées pour contrôler l'interaction entre les différents objets et le formulaire dans son ensemble.

Comme les formulaires peuvent être utilisés de nombreuses manières, il est utile de contrôler ce qui se passe lorsque votre formulaire est en cours d'utilisation. Pour cela, 4D met à votre disposition un grand nombre **d'événements formulaires**. Ces événements vous indiquent précisément ce qui se produit dans le formulaire. Chaque type d'événement (par exemple un clic, un double-clic, une touche du clavier enfoncée...) active ou désactive l'exécution de la méthode formulaire ainsi que les méthodes des objets du formulaire.

Pour plus d'informations sur les formulaires, les objets, les événements et les méthodes, reportez-vous à la description de la commande Evenement formulaire.

Réguler le fonctionnement de votre base à l'aide des méthodes table/triggers

Un trigger est une méthode associée à une table, c'est la raison pour laquelle on peut également parler de méthode table. Les triggers sont automatiquement appelés par le moteur de base de données de 4D à chaque fois qu'une action (ajout, suppression, modification et chargement) est effectuée sur les enregistrements d'une table. Les triggers peuvent empêcher que des opérations interdites soient exécutées avec les enregistrements de votre base. Par exemple, dans un système de facturation, vous ne voulez pas qu'un utilisateur puisse créer des factures sans avoir spécifié le client à qui elle est destinée. Les triggers sont des outils puissants de contrôle des opérations possibles avec les tables, ainsi que de prévention des risques de pertes accidentelles de données. Vous pouvez écrire des triggers très simples, puis les rendre de plus en plus sophistiqués.

Pour plus d'informations sur les triggers, reportez-vous à la section Présentation des triggers.

Utiliser des méthodes projet dans toute la base

A la différence des triggers, des méthodes formulaire et des méthodes objet, qui sont associées à des tables, des formulaires ou des objets actifs particuliers, les méthodes projet sont accessibles depuis n'importe quelle partie de votre base. Les méthodes projet sont réutilisables et peuvent être appelées par d'autres méthodes. Vous pouvez ainsi effectuer plusieurs fois une opération similaire sans devoir écrire plusieurs méthodes.
Vous pouvez appeler des méthodes projet depuis n'importe quelle partie de la base — autres méthodes projet, méthodes objet, méthodes formulaire... Lorsque vous appelez une méthode projet, elle s'exécute comme si elle avait été écrite à l'endroit d'où elle a été appelée. Les méthodes projet appelées depuis d'autres procédures sont appelées sous-routines.
Il y a un autre moyen d'utiliser les méthodes projet : les associer à des commandes de menu. Lorsque vous associez une méthode projet à une commande de menu, la méthode est exécutée lorsque la commande est sélectionnée par l'utilisateur. Vous pouvez considérer la commande de menu comme un appel à une méthode projet.

Gérer les sessions de travail avec les méthodes base

De la même manière que les méthodes objet et formulaire sont exécutées lorsque des événements se produisent dans un formulaire, il existe des méthodes associées à la base qui sont exécutées lorsqu'un événement de type "session de travail" se produit : ce sont les **méthodes base**.

Par exemple, à chaque fois que vous ouvrez une base de données, vous voulez initialiser certaines variables qui seront utilisées pendant toute la session de travail ; pour cela, vous pouvez initialiser vos variables dans la méthode base Sur ouverture, qui est automatiquement exécutée par 4D lorsque vous ouvrez la base.

Pour plus d'informations sur les méthode base, reportez-vous à la section Présentation des méthodes base.

Développer votre base de données

La phase de développement est celle pendant laquelle vous personnalisez votre base à l'aide du langage et des autres fonctions intégrées.

En créant simplement une base, vous avez déjà franchi les premières étapes de l'utilisation du langage. Chaque élément d'une base — les tables et les champs, les formulaires et leurs objets — est automatiquement relié au langage. Le langage de 4D les "reconnaît" tous.

Votre première utilisation du langage pourrait être de créer une méthode objet ou formulaire pour contrôler la saisie de données. Par la suite, vous pourrez définir une méthode formulaire pour gérer l'affichage. Lorsque la base deviendra plus complexe, vous ajouterez une barre de menus avec des méthodes projet pour la personnaliser entièrement.

Comme tous les autres aspects de 4D, le développement est une phase très souple. Il n'y a aucun cheminement précis à suivre — vous pouvez développer de la manière qui vous semble la plus pratique. Bien entendu, cette phase comprend des étapes générales.

• L'implémentation : définition de la structure et des fonctionnalités de la base en mode Développement.
• Le test : réalisation d'essais de la base et de test de chaque élément personnalisé en mode Test application.
• L'utilisation : lorsque la base est entièrement personnalisée, exécution en mode Application.
• Les corrections : si des erreurs sont détectées, retour au mode Développement pour les corriger.

Des outils particuliers d'aide au développement, n'apparaissant que lorsque c'est nécessaire, sont intégrés à 4D. A mesure que vous utiliserez le langage de manière de plus en plus intensive, vous vous apercevrez que ces outils facilitent grandement le développement. Par exemple, l'éditeur de méthodes traite automatiquement les éventuelles erreurs de saisie et formate votre travail à la volée ; l'interpréteur (le moteur qui exécute le langage) intercepte les erreurs de syntaxe et vous les désigne ; enfin, le débogueur vous permet de contrôler très précisément l'exécution de vos méthodes afin de détecter toute erreur de développement.

Construire des applications

Vous connaissez maintenant les opérations que l'on peut réaliser avec une base de données — saisie, recherches, tris et créations d'états... Vous avez pu effectuer ces actions depuis le mode Développement, à l'aide des menus et des éditeurs standard.

Lorsque vous utilisez une base de données, vous répétez certaines séquences de tâches. Par exemple, dans une base de contacts personnels, vous pouvez rechercher des contacts, les trier par nom, puis imprimer un état à chaque fois que des informations ont été modifiées. Ces opérations ne sont pas difficiles à réaliser, mais elles peuvent prendre beaucoup de temps lorsqu'il faut les faire 20 fois. De plus, si vous n'avez pas utilisé la base pendant deux semaines, il se peut que vous ayez oublié certaines étapes nécessaires à la création d'un état.
Dans les méthodes, les étapes s'enchaînent les unes après les autres. Ainsi, une seule commande effectue automatiquement toutes les tâches qui lui sont liées. Par conséquent, vous n'avez pas à vous préoccuper des opérations particulières à réaliser.

Dans vos applications, les menus créés permettent d'effectuer des tâches répondant précisément aux besoins de la personne qui utilise la base. Une application se compose de tous les éléments de votre base : la structure, les formulaires, les différentes méthodes, les menus et les mots de passe.

Vous pouvez compiler vos bases et créer des applications Windows et Mac OS autonomes. La compilation des bases de données accélère l'exécution du langage, protège les bases, et vous permet de créer des applications totalement indépendantes. Le compilateur intégré vérifie également la syntaxe ainsi que les types des variables dans les méthodes, et contrôle donc la cohérence de base.

Les applications que vous créez peuvent aller du plus simple (un menu unique permettant de saisir des noms et d'imprimer un état) au plus complexe (un système de facturation ou de gestion des stocks). Les types d'utilisation des applications sont illimités. Généralement, une application grandit progressivement depuis une base de données en mode Développement jusqu'à un logiciel entièrement contrôlé par des menus et des formulaires personnalisés.

Pour en savoir plus...

• Le développement d'applications peut être aussi simple ou aussi complexe que vous le voulez. Pour plus d'informations sur le processus de construction d'une application, reportez-vous à la section Construire une application 4D.
• Si vous débutez avec 4D, reportez-vous aux sections **Présentation du langage** pour découvrir les principes de fonctionnement du langage 4D. Commencez par la section Introduction au langage 4D.

Une application 4D est une base de données conçue pour répondre spécifiquement à des besoins précis. Une application comporte une interface utilisateur destinée à faciliter son utilisation. Toutes les fonctions qu'elle propose sont directement liées (et se limitent) à son champ d'action. 4D vous permet de créer des applications de manière plus confortable et plus aisée que si vous utilisiez des langages traditionnels.
Parmi la grande variété d'applications que 4D peut créer, citons :

- un système de facturation,
- un système de gestion des stocks,
- un système de comptabilité,
- un système de paie,
- un gestion des ressources humaines,
- un système de traitement des prospects,
- une base de données accessible par Internet ou Intranet.

Il est parfaitement envisageable qu'une seule application 4D gère tous ces systèmes. Ce type d'applications correspond à une utilisation plutôt traditionnelle des bases de données. De surcroît, les outils proposés par 4D vous permettent de créer des applications originales, telles que, par exemple :

- un système de gestion documentaire,
- un système graphique de gestion d'images,
- une application de publication de catalogues,
- un système de contrôle et de commande d'un dispositif externe,
- un système de messagerie électronique (E-mail),
- un système multi-utilisateurs de gestion de planning,
- un catalogue recensant les éléments d'une collection de vidéos ou de disques.

Typiquement, une application peut commencer par être une simple base de données exploitée en mode Développement. Cette base "se transforme" en application à mesure qu'elle est personnalisée. La caractéristique majeure d'une application est la dissimulation à l'utilisateur des systèmes internes de gestion des fonctions de la base. La gestion de la base est automatisée, et l'utilisateur se sert de menus personnalisés pour exécuter des tâches spécifiques.

Lorsqu'une base 4D est exploitée en mode Développement, vous devez connaître les étapes à atteindre pour obtenir le résultat recherché. Dans une application 4D, c'est le mode Application qui est utilisé. Dans ce mode, vous devez gérer vous-même toutes les fonctions qui sont automatiques dans le mode Développement, c'est-à-dire :

• Navigation parmi les tables : la Liste des tables, la commande **Dernière tables utilisées** ou les boutons de navigation ne sont pas accessibles à l'utilisateur. Vous pouvez utiliser les commandes de menus et les méthodes pour contrôler la navigation parmi les tables.
• Menus : en mode Application, seuls les menus Fichier (comportant la commande Quitter), Edition, Mode et Aide (ainsi que le menu application sous Mac OS) sont affichés par défaut. Si l'application nécessite d'autres menus, vous devez les créer vous-même et les gérer à l'aide de méthodes 4D ou d'actions standard.
• Editeurs : les éditeurs, tels que les éditeurs de recherches et de tris, ne sont plus automatiquement accessibles en mode Application. Si vous voulez qu'ils restent disponibles, vous devez les appeler en utilisant les commandes du langage.

Les paragraphes suivants fournissent des exemples d'automatisation de l'utilisation d'une base à l'aide du langage.

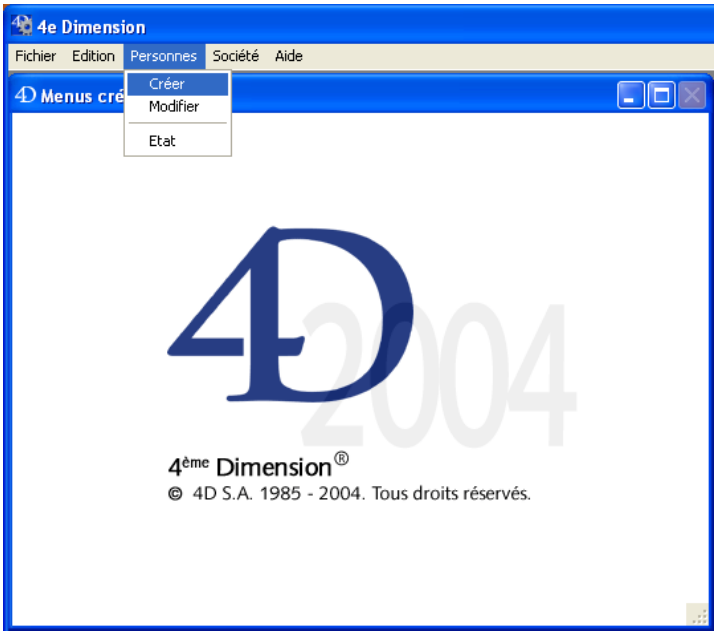
Mode Application : un exemple

Les menus personnalisés sont les éléments d'interface élémentaires d'une application. La création de menus est très simple — il suffit d'associer une méthode ou une action standard à chaque commande de menu dans l'éditeur de menus.

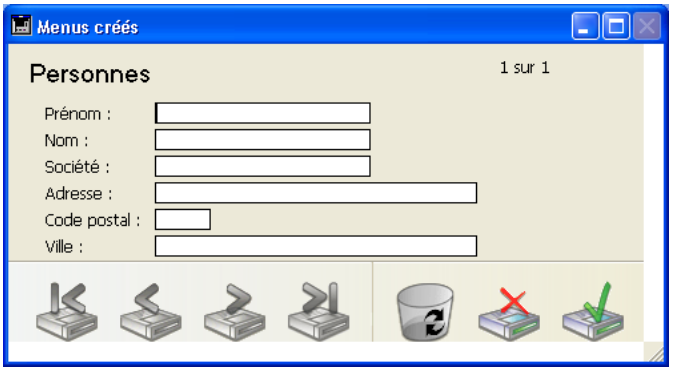
Les menus personnalisés facilitent l'apprentissage et l'utilisation d'une base de données.

Le paragraphe ci-dessous décrit le point de vue de l'utilisateur lorsqu'il choisit une commande de menu. Le paragraphe suivant détaille ensuite ce qui se produit réellement au cœur l'application ainsi que le travail de conception qui a été effectué. Bien que l'exemple soit simple, il apparaît clairement que la base est plus facile à apprendre et à utiliser. L'utilisateur n'a accès qu'aux éléments correspondant à ses besoins plutôt qu'aux outils "génériques" et aux commandes de menu du mode Développement.

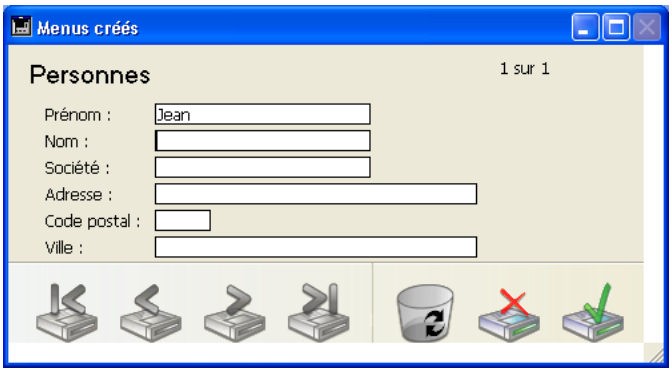
Le point de vue de l'utilisateur
L'utilisateur sélectionne une commande de menu personnalisé appelée **Créer** pour ajouter une nouvelle personne dans la base de données.



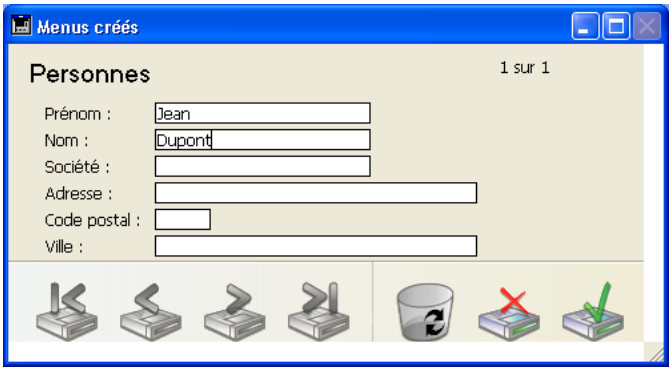
Le formulaire entrée de la table [Personnes] s'affiche.



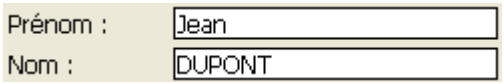
L'utilisateur saisit le prénom de la personne et appuie sur la touche **Tabulation** pour passer au champ suivant.



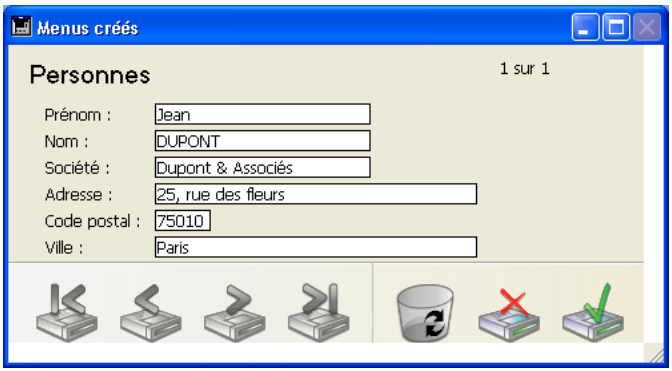
L'utilisateur saisit le nom de la personne.



L'utilisateur appuie sur la touche **Tabulation** pour passer au champ suivant : le nom est converti en lettres majuscules.



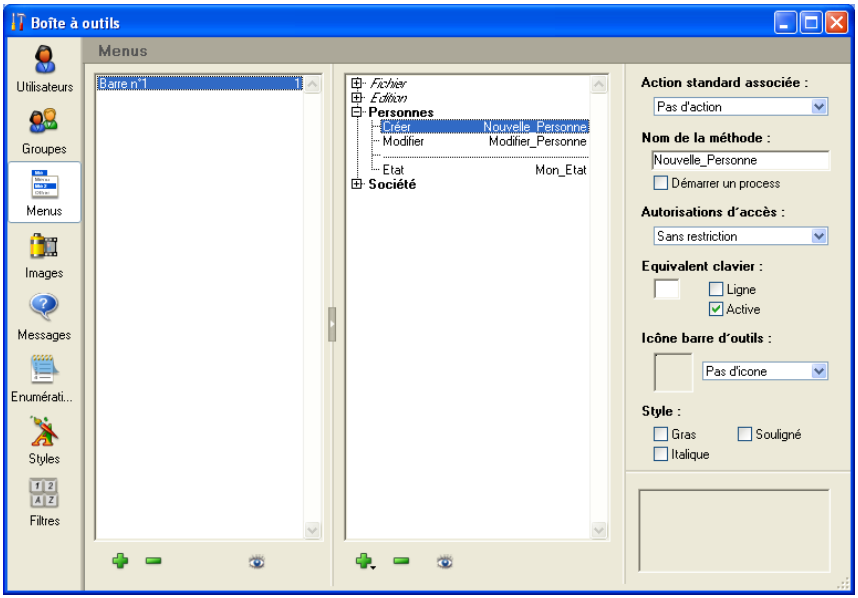
Une fois la saisie terminée, l'utilisateur clique sur le bouton de validation (généralement le dernier dans la barre de boutons).



Un autre enregistrement vide s'affiche. L'utilisateur clique sur le bouton **Annuler** (celui qui comporte une croix) pour terminer la "boucle de saisie des données". L'utilisateur retourne à l'écran principal.

Les coulisses

La barre de menus a été créée en mode Développement, à l'aide de l'éditeur de menus.



La commande de menu **Créer** est associée à une méthode projet appelée Nouvelle Personne. Cette méthode a été créée, en mode Développement, dans l'éditeur de méthodes. Lorsque l'utilisateur sélectionne cette commande de menu, la méthode Nouvelle personne s'exécute.

La boucle Repeter...Jusque à l'intérieur de laquelle se trouve la commande AJOUTER ENREGISTREMENT provoque exactement les mêmes effets que la commande de menu **Nouvel enregistrement** en mode Développement. Elle affiche le formulaire entrée courant, de manière à ce que l'utilisateur puisse saisir un nouvel enregistrement. Lorsque l'utilisateur valide l'enregistrement, un autre enregistrement vide apparaît. Cette boucle AJOUTER ENREGISTREMENT continue de s'exécuter jusqu'à ce que l'utilisateur clique sur le bouton **Annuler**.

Lorsque l'utilisateur remplit un enregistrement, les actions suivantes sont déclenchées :
• Comme il n'y a pas de méthode objet associée au champ Prénom, rien ne s'exécute.
• Une méthode est associée au champ Nom. Cette méthode a été créée, en mode Développement, à l'aide des éditeurs de formulaires et de méthodes. Elle exécute l'instruction suivante :

```
Nom:=Majusc(Nom)
```

Cette ligne convertit le champ "Nom" en caractères majuscules.

Une fois qu'un enregistrement a été saisi, lorsque l'utilisateur clique sur le bouton d'annulation dans le formulaire suivant, la variable système OK prend la valeur zéro, ce qui constitue la condition d'arrêt de l'exécution de la boucle AJOUTER ENREGISTREMENT.

Comme il n'y a pas d'autres instructions à exécuter, la méthode Nouvelle Personne stoppe son exécution et rend la main à la barre de menus principale.

Comparer une application 4D avec le mode Développement

Comparons la manière dont une même tâche est effectuée en mode Développement et à l'aide du langage.

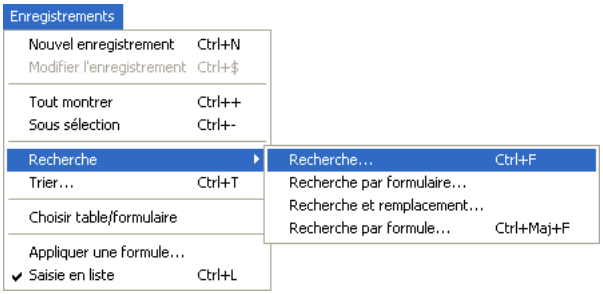
Examinons une opération courante :
• Sélectionner un groupe d'enregistrements,
• Le trier,
• Imprimer un état.

Le paragraphe ci-dessous, "Exploiter une base en mode Développement", traite de la réalisation de ces tâches à partir du mode Développement. Le paragraphe suivant, "Exploiter une application 4D avec les éditeurs intégrés", traite de la réalisation des mêmes tâches à partir du mode Application.

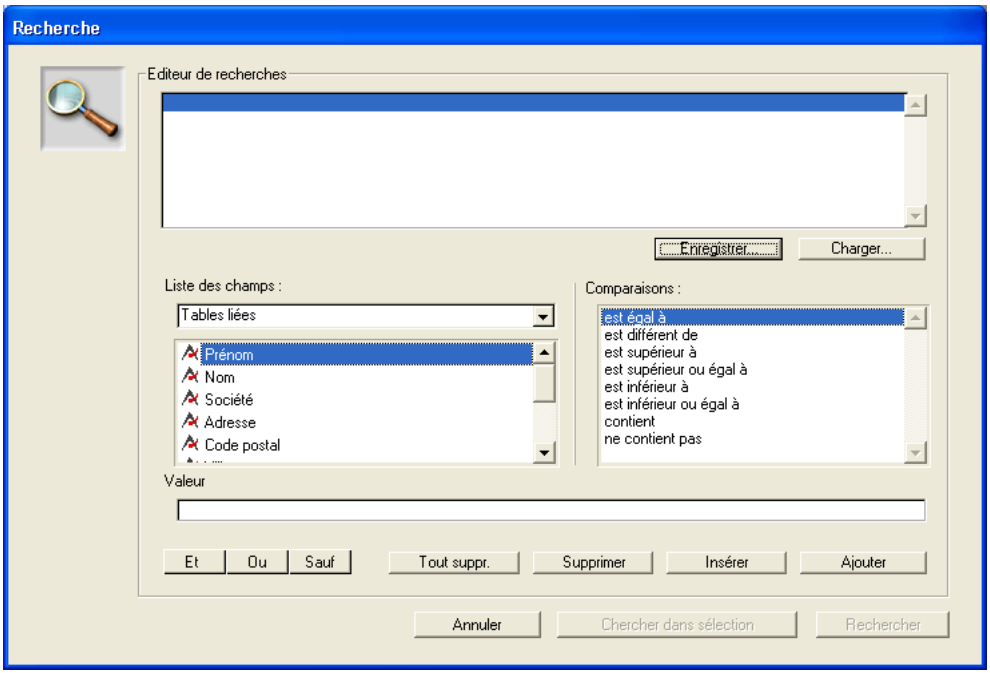
Notez que, bien que les mêmes opérations sont effectuées dans les deux cas, les étapes dans le second paragraphe sont automatisées par programmation.

Exploiter une base en mode Développement

L'utilisateur choisit la commande **Rechercher>Rechercher...** dans le menu **Enregistrements**.



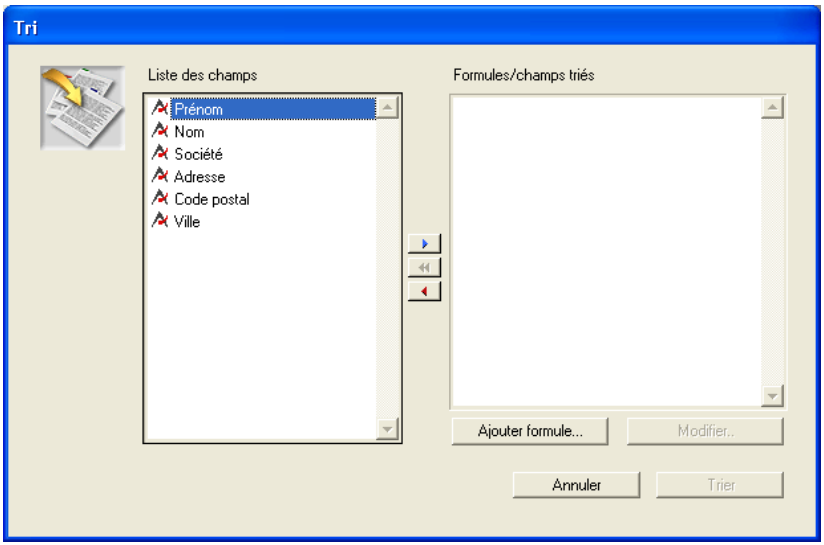
L'éditeur de recherches s'affiche :



L'utilisateur définit ses critères de recherche et clique sur le bouton **Rechercher**. La recherche s'effectue.

L'utilisateur choisit la commande **Trier...** dans le menu **Enregistrements**.

L'éditeur de tris s'affiche.



L'utilisateur définit ses critères de tri et clique sur le bouton **Trier**. Le tri est effectué.

Puis, pour imprimer les enregistrements, les étapes suivantes sont nécessaires :
• L'utilisateur choisit la commande **Imprimer** dans le menu **Fichier**.
La boîte de dialogue de choix du formulaire d'impression s'affiche (l'utilisateur doit savoir quel formulaire choisir !).
• Les boîtes de dialogue de l'imprimante sélectionnée s'affichent. L'utilisateur fixe ses paramètres, puis l'état est imprimé.

Exploiter une application 4D avec les éditeurs intégrés

Examinons maintenant comment ces opérations peuvent être effectuées en mode Application.

L'utilisateur choisit la commande **Etat** dans le menu **Personnes**.

Même à ce stade, l'utilisation d'une application apparaît plus facile. L'utilisateur n'a pas besoin de savoir qu'il faut commencer par effectuer une recherche.

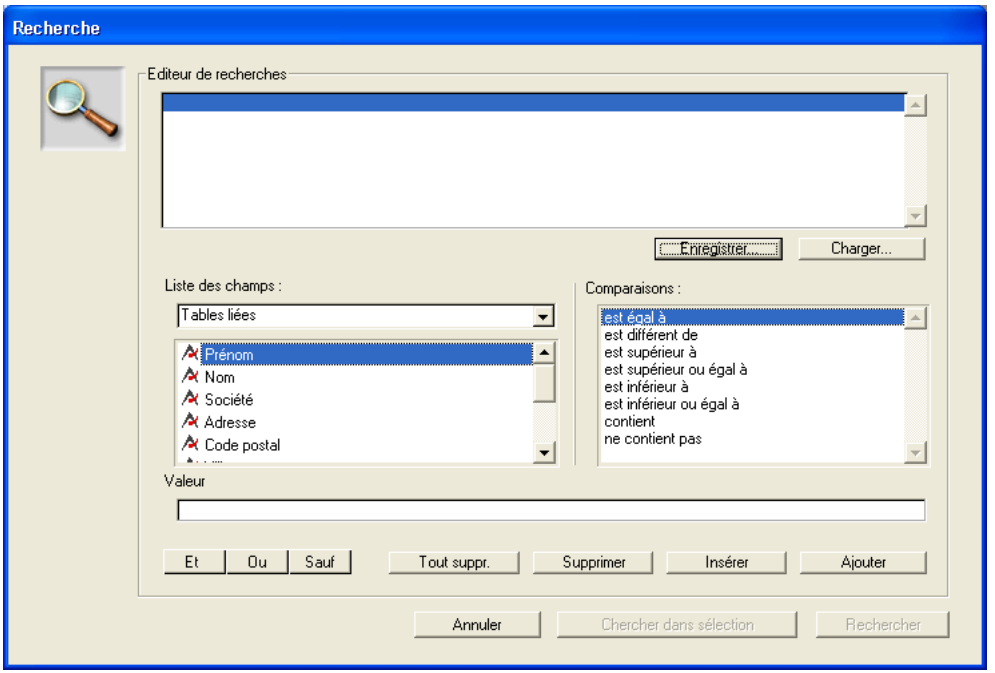
Une méthode appelée Mon Etat est associée à la commande de menu. Elle se présente ainsi :

```
CHERCHER ([Personnes])
TRIER ([Personnes])
FORMULAIRE SORTIE ([Personnes]; "Etat")
IMPRIMER SELECTION ([Personnes])
```

La première ligne est exécutée :

```
CHERCHER ([Personnes])
```

L'éditeur de recherches s'affiche.



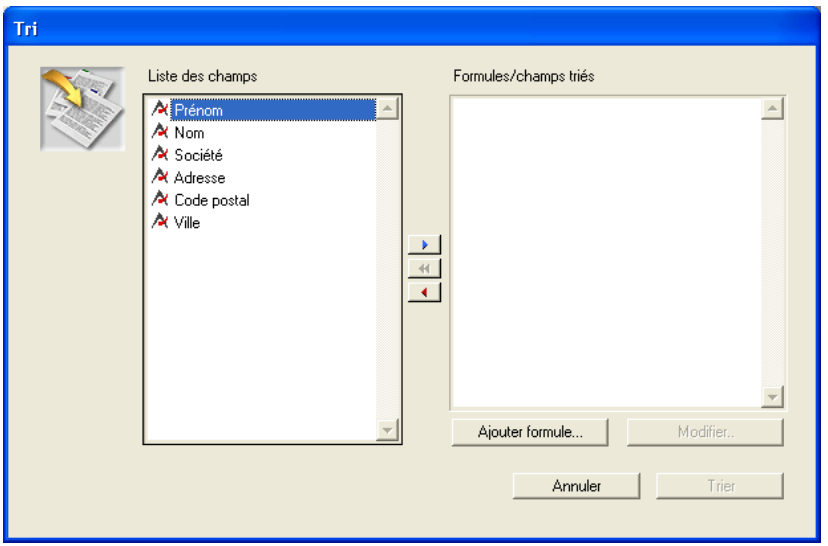
L'utilisateur définit ses critères de recherche et clique sur le bouton **Rechercher**. La recherche s'effectue.

La deuxième ligne de la méthode Mon Etat est exécutée :

```
TRIER ([Personnes])
```

Vous notez que l'utilisateur n'avait pas besoin de savoir que le tri était la deuxième étape.

La boîte de dialogue de tri s'affiche.



L'utilisateur définit ses critères de tri et clique sur le bouton **Trier**. Le tri est effectué.

La troisième ligne de Mon Etat est exécutée :

FORMULAIRE SORTIE ([Personnes]; "Etat")

Une fois de plus, il n'est pas nécessaire que l'utilisateur sache ce qu'il faut faire ensuite. Le cheminement est déjà tracé.

La dernière ligne de la méthode Mon Etat est exécutée :

IMPRIMER SELECTION ([Personnes])

Les boîtes de dialogue de l'imprimante sélectionnée s'affichent. L'utilisateur fixe ses paramètres, puis l'état est imprimé.

Les mêmes commandes que celles qui ont été employées dans la comparaison précédente peuvent être utilisées pour automatiser encore plus la base de données.

Examinons la nouvelle version de la méthode Mon Etat.

L'utilisateur choisit **Etat** dans le menu **Personnes**. La méthode appelé Mon Etat2 est associée à la commande de menu :

```
CHERCHER ([Personnes]; [Personnes]Entreprise = "Dupont & Associés")
TRIER ([Personnes]; [Personnes]Nom; > ; [Personnes]Prénom; >)
FORMULAIRE SORTIE([Personnes]; "Etat")
IMPRIMER SELECTION ([Personnes]; *)
```

La première ligne est exécutée :

```
CHERCHER ([Personnes]; [Personnes]Entreprise = "Dupont & Associés")
```

L'éditeur de recherches ne s'affiche pas. Au lieu de cela, la recherche est définie et effectuée par la commande CHERCHER. L'utilisateur n'a rien à faire.

La deuxième ligne est exécutée :

```
TRIER ([Personnes]; [Personnes]Nom; > ; [Personnes]Prénom; >)
```

La boîte de dialogue de tri n'est pas affichée, et le tri est immédiatement effectué. Une fois encore, aucune intervention de l'utilisateur n'est nécessaire.

Les dernières lignes de la méthode Mon Etat2 sont exécutées :

```
FORMULAIRE SORTIE([Personnes]; "Etat")
IMPRIMER SELECTION ([Personnes]; *)
```

Les boîtes de dialogue standard d'impression ne sont pas affichées. La commande IMPRIMER SELECTION comporte en effet le paramètre optionnel astérisque (*), ce qui lui indique d'utiliser les paramètres d'impression en vigueur au moment où le formulaire d'état a été créé. L'état est imprimé.

Les avantages de cette automatisation accrue sont les suivants :
• La recherche est effectuée automatiquement : cela évite que les utilisateurs sélectionnent de mauvais critères lorsqu'ils construisent leur recherche.
• Le tri est effectué automatiquement : cela évite que les utilisateurs sélectionnent de mauvais critères lorsqu'ils construisent leur tri.

• L'impression est effectuée automatiquement : cela évite que les utilisateurs sélectionnent un formulaire inadapté.
• Vous évitez à l'utilisateur d'avoir à définir des options dans les trois boîtes de dialogue.

Aides au développement d'applications 4D

A mesure que vous progresserez dans votre développement d'une application 4D, vous allez découvrir de nombreuses fonctionnalités que vous n'aviez pas vues lorsque vous avez commencé.
Mais vous pouvez aller encore plus loin, en ajoutant des outils et des plug-ins dans votre environnement 4D.

Plug-ins 4D

4D fournit plusieurs plug-ins permettant d'augmenter les capacités de vos applications :

• **4D Write** : Traitement de textes
• **4D Draw** : Créateur de documents graphiques de type CAD
• **4D View** : Tableur et éditeur de listes.
• **4D Internet Commands** (intégré) : Utilitaires de communication via Internet

• **4D ODBC Pro** : Connectivité bas niveau via ODBC
• **4D for OCI** : Connectivité avec ORACLE Call Interface
• **4D Open for Java** : Connectivité avec les applications JAVA
• **4D Open for 4D** : Connectivité de 4D à 4D pour construire des systèmes 4D d'information distribuée.

Pour plus d'informations, contactez 4D ou un Partenaire 4D, ou visitez notre site Web :

http://www.4D.fr

La communauté 4D et les produits des partenaires 4D

Il existe une communauté internationale très active organisée autour de 4D, composée de groupes d'utilisateurs, de forums électroniques et de partenaires 4D, qui développe des **produits liés à 4D**.

Vous pouvez vous inscrire sur le forum public d'aide et d'échange de 4D à l'adresse suivante :

http://forums.4D.fr

La communauté 4D vous fournira de nombreux conseils et solutions, ainsi que des produits tiers vous permettant d'augmenter votre productivité. Vous aurez accès à une foule d'informations et d'astuces qui vous feront économiser votre temps et votre énergie.

2

Présentation du langage

Le langage de 4D est constitué de différents éléments, vous permettant d'effectuer de multiples opérations et de gérer vos données.

• **Types de données** : Catégories de données stockées dans une base. Ce point est traité dans le paragraphe suivant. Pour une description détaillée, référez-vous à la section Types de données.
• **Variables** : Adresses de stockage temporaires de données en mémoire. Pour une description détaillée, référez-vous à la section Variables.
• **Opérateurs** : Symboles effectuant un calcul entre deux valeurs. Ce point est traité dans les paragraphes suivants. Pour une description détaillée, référez-vous à la section Opérateurs et à ses sous-sections.
• **Expressions** : Combinaisons d'éléments du langage ayant pour résultat le renvoi d'une valeur. Ce point est traité dans les paragraphes suivants.
• **Commandes** : Instructions intégrées effectuant une opération. Toutes les commandes de 4D (par exemple AJOUTER ENREGISTREMENT) sont décrites dans ce manuel, groupées par thème. Lorsque c'est nécessaire, les thèmes comprennent une section d'introduction. Vous pouvez utiliser des Plug-ins 4D pour ajouter de nouvelles commandes à votre environnement de développement 4D. Par exemple, une fois que vous avez installé le plug-in 4D Write dans votre base de données, vous pouvez utiliser les commandes de 4D Write pour créer et manipuler des fichiers de traitement de texte.
• **Méthodes** : Instructions que vous écrivez à l'aide de tous les éléments du langage décrits ci-dessus. Pour une description détaillée, référez-vous à la section Méthodes et à ses sous-sections.

Cette section présente les Types de données, les Opérateurs et les Expressions. Pour une description d'un autre élément, reportez-vous aux sections précédemment citées.

De plus :
• Les éléments du langage tels que les variables sont identifiés par leur nom. Pour une description détaillée des règles de définition des noms d'objets, reportez-vous à la section Nommer les objets du langage 4D.
• Pour plus d'informations sur les variables tableaux, reportez-vous à la section Présentation des tableaux.
• Pour plus d'informations sur les variables BLOB, reportez-vous à la section Commandes du thème BLOB.
• Si vous avez l'intention de compiler votre base, reportez-vous à la section Commandes du thème Compilateur ainsi qu'au manuel *Mode Développement* de 4D.

Types de données

De nombreux types de données peuvent être stockés dans une base 4D. Le langage distingue sept types de données élémentaires : chaîne, numérique, date, heure, booléen, image et pointeur.

• **Chaîne** : Une suite de caractères, telles que “Bonjour à tous”. Les champs et variables alpha et texte sont des données de type Chaîne.
• **Numérique** : Nombres, tels que 2 ou 1 000,67. Les champs et variables entier, entier long et numérique (aussi appelé réel) sont des données de type Numérique.
• **Date** : Dates telles que 20/11/97. Les champs et variables date sont des données de type Date.
• **Heures** : Heures, c’est-à-dire heures, minutes et secondes, telles que 21:00:00 ou 4:35:30 de l'après-midi. Les champs et variables heure sont des données de type Heure.
• **Booléen** : Valeurs logiques de VRAI ou FAUX. Les champs et variables booléens sont des données de type Booléen.
• **Image** : Les champs et variables image sont des données de type Image.
• **Pointeur** : Type spécial de données, utilisé en programmation avancée. Les variables pointeur sont des données de type Pointeur. Il n’y a pas de type de champ correspondant.

Vous constatez que, dans cette liste de types de données, les types chaîne et numérique sont associés à plus d’un type de champ. Lorsque des données sont placées dans un champ, le langage les convertit automatiquement dans le type du champ. Par exemple, si un champ de type entier est utilisé, les valeurs qu’il contient sont automatiquement traitées en tant que numériques. En d’autres termes, vous n’avez pas à vous préoccuper du mélange de champs de types semblables lorsque vous programmez avec 4D ; le langage le gère pour vous.

Cependant, il est important, lorsque vous utilisez le langage, de ne pas mélanger les types de données différents. Tout comme il est absurde de stocker la valeur “ABC” dans un champ de type Date, il est absurde de donner la valeur “ABC” à une variable utilisée pour des dates. Dans la plupart des cas, 4D est très tolérant et tentera d’utiliser de manière logique ce que vous faites. Par exemple, si vous additionnez un nombre x et une date, 4D déduira que vous voulez ajouter x jours à la date, mais si vous tentez d’ajouter une chaîne à une date, 4D vous préviendra que cette opération est impossible.

Certains cas nécessitent que vous stockiez des données dans un type et que vous les utilisiez dans un autre. Le langage contient un ensemble complet de commandes vous permettant de convertir des types de données vers d’autres types. Par exemple, si vous voulez créer un numéro de matricule commençant par des chiffres et se terminant par des lettres, vous pouvez écrire :

```
[Produits]Matricule:=Chaine(Numéro)+"abc"
```

où, si Numéro vaut 17, [Produits]Matricule prendra la valeur “17abc”.

Les types de données sont détaillés dans la section Types de données.

Opérateurs

Lorsque vous programmez avec 4D, il est rare que vous ayez simplement besoin de données “brutes”. Le plus souvent, il sera nécessaire de traiter ces données d'une manière ou d'une autre. Vous effectuez ces calculs avec des **opérateurs**. Les opérateurs, en général, prennent deux valeurs et effectuent avec elles une opération dont le résultat est une troisième valeur. Vous connaissez déjà la plupart des opérateurs. Par exemple, 1 + 2 utilise l’opérateur d’addition (ou signe plus) pour faire la somme de deux nombres, et le résultat est 3. Le tableau ci-dessous présente quelques opérateurs courants :

Opérateur	Opération	Exemple
+	Addition	1 + 2 ce qui fait 3
–	Soustraction	3 – 2 ce qui fait 1
*	Multiplication	2 * 3 ce qui fait 6
/	Division	6 / 2 ce qui fait 3

Les opérateurs numériques ne représentent qu’un seul des différents types d’opérateurs disponibles. Comme 4D traite de multiples types de données, tels que des nombres, des dates ou des images, vous disposez d’opérateurs particuliers effectuant des opérations sur ces données.

Souvent, les mêmes symboles sont utilisés pour des opérations différentes, en fonction du type de données traitées. Par exemple, le signe (+) peut effectuer diverses opérations, comme le montre le tableau suivant :

Type de données	Opération	Exemple
Numérique	Addition	1 + 2 ajoute les nombres, le résultat est 3
Chaîne	Concaténation	“Bonjour ” + “à tous” concatène (met bout à bout) les chaînes, le résultat est “Bonjour à tous”
Date et Numérique	Addition de date	!1/1/1997! + 20 ajoute 20 jours à la date 1 janvier 1997, le résultat est la date 21 janvier 1997

Les opérateurs sont détaillés dans la section Opérateurs et ses sous-sections.

Expressions

Pour parler simplement, les expressions retournent une valeur. En fait, lorsque vous programmez avec 4D, vous utilisez tout le temps des expressions et vous avez tendance à les manipuler uniquement à travers la valeur qu'elles représentent. Les expressions sont aussi appelées formules.

Les expressions peuvent être constituées de presque tous les éléments du langage : commandes, opérateurs, variables et champs. Vous utilisez des expressions pour écrire des lignes de code, qui sont à leur tour utilisées pour construire des méthodes. Des expressions sont employées à chaque fois que le langage de 4D a besoin de connaître la valeur d'une donnée.

Les expressions sont rarement "indépendantes". Il n'y a que peu d'endroits dans 4D où une expression peut être utilisée en tant que telle : dans la boîte de dialogue de Recherche par formule, dans la fenêtre du Débogueur où la valeur des expressions peut être évaluée, dans la boîte de dialogue Appliquer une formule, et dans l'éditeur d'états semi-automatiques en tant que formule dans une colonne.

Une expression peut être simplement une constante, telle que le chiffre 4 ou la chaîne "Bonjour". Comme son nom l'indique, la valeur d'une constante ne change jamais. C'est lorsqu'elles contiennent des opérateurs que les expressions commencent à devenir intéressantes. Dans les paragraphes précédents, vous avez déjà pu voir des expressions utilisant des opérateurs. Par exemple, 4 + 2 est une expression qui utilise l'opérateur d'addition pour additionner deux nombres, et dont le résultat est 6.

Vous vous référez à une expression par le biais du type de données qu'elle retourne. Il existe sept types d'expressions :

- Expression chaîne,
- Expression numérique (aussi appelée nombre),
- Expression date,
- Expression heure,
- Expression booléenne,
- Expression image,
- Expression pointeur.

Le tableau suivant fournit un exemple pour chacun des sept types d'expressions.

Expression	Type	Description
"Bonjour"	Chaîne	Le mot Bonjour est une constante chaîne, signalée par les guillemets.
"Bonjour " + "à tous"	Chaîne	Deux chaînes, "Bonjour " et "à tous", sont mises bout à bout (concaténées) à l'aide de l'opérateur de concaténation de chaînes (+). La chaîne "Bonjour à tous" est retournée.
"M. " + [Amis]Nom	Chaîne	Deux chaînes sont concaténées : la chaîne "M. " et la valeur courante du champ Nom de la table Amis. Si le champ contient "Dupont", l'expression retourne "M. Dupont".
Majusc ("dupont")	Chaîne	Cette expression utilise "Majusc", une commande du langage, pour convertir la chaîne "dupont" en majuscules. Elle retourne "DUPONT".
4	Numérique	C'est une constante numérique, 4.
4 * 2	Numérique	Deux nombres, 4 et 2, sont multipliés à l'aide de l'opérateur de multiplication (*). Le résultat est le nombre 8.
MonBouton	Numérique	C'est le nom d'un bouton. Il retourne la valeur courante du bouton : 1 s'il y a eu un clic sur le bouton, 0 sinon.
!25/1/97!	Date	C'est une constante date pour la date 25/01/97 (25 janvier 1997). Notez l'emploi des points d'exclamation pour indiquer une constante date.
Date du jour + 30	Date	C'est une expression de type Date qui utilise la fonction Date du jour pour récupérer la date courante. Elle ajoute 30 jours à la date d'aujourd'hui et retourne la nouvelle date.
?8:05:30?	Heure	C'est une constante heure qui représente 8 heures, 5 minutes, et 30 secondes.
?2:03:04? + ?1:02:03?	Heure	Cette expression ajoute une heure à une autre et retourne l'heure 3:05:07.
Vrai	Booléen	Cette commande retourne la valeur booléenne VRAI.

10 # 20	Booléen	C'est une comparaison logique entre deux nombres. Le symbole (#) signifie “est différent de”. Comme 10 “est différent de” 20, l'expression retourne VRAI.
“ABC” = “XYZ”	Booléen	C'est une comparaison logique entre deux chaînes. Elles sont différentes, donc l'expression retourne FAUX.
MonImage + 50	Image	Cette expression considère l'image placée dans MonImage, la déplace de 50 pixels vers la droite, et retourne l'image résultante.
->[Amis]Nom	Pointeur	Cette expression retourne un pointeur vers le champ [Amis]Nom.
Table (1)	Pointeur	C'est une commande qui retourne un pointeur vers la première table.

Référence

Constantes, Méthodes, Opérateurs, Pointeurs, Présentation des tableaux, Types de données, Variables.

Les champs, variables et expressions de 4D ont un type représentant les données qu'ils contiennent. 4D accepte le typage de ces éléments en fonction du tableau suivant :

Types de données	Champ	Variable	Expression
Chaîne (cf. note 1)	Oui	Oui	Oui
Numérique (cf. note 2)	Oui	Oui	Oui
Date	Oui	Oui	Oui
Heure	Oui	Oui	Oui
Booléen	Oui	Oui	Oui
Image	Oui	Oui	Oui
Pointeur	Non	Oui	Oui
BLOB (cf. note 3)	Oui	Oui	Non
Tableau (cf. note 4)	Non	Oui	Non
Entier 64 bits (cf. note 5)	Oui	Non	Non
Float (cf. note 5)	Oui	Non	Non
Indéfini	Non	Oui	Oui

Notes
(1) Une chaîne peut être un champ alphanumérique, une variable de longueur fixe, ou encore une variable ou un champ de type Texte.
(2) Un numérique peut être une variable ou un champ de type Réel (Numérique), Entier et Entier long.
(3) BLOB est l'abréviation de Binary Large OBject. Pour plus d'informations sur les BLOBs, reportez-vous à la section Commandes du thème BLOB.
(4) Les tableaux peuvent être de tout type. Pour plus d'informations, reportez-vous à la section Présentation des tableaux.
(5) Les types Entier 64 bits et Float sont gérés uniquement via le SQL. Il est déconseillé de les manipuler via le langage 4D, car dans ce cas ils sont convertis en Réels, ce qui peut entraîner des pertes de précision.

Chaîne

Chaîne est un terme générique utilisé pour :
• les variables ou champs de type alphanumérique
• les variables ou champs de type Texte
• toute expression de type Alpha ou Texte

Une chaîne se compose de caractères. La gestion des chaînes de caractères varie suivant que 4D est exécuté en mode Unicode ou en mode non Unicode (mode compatibilité). Ce mode est défini via les Préférences de l'application (cf. section Codes ASCII).

Mode Unicode

• Un champ alphanumérique peut contenir de 0 à 255 caractères (la limite est fixée lors de la définition du champ).
• Un champ, une variable ou une expression de type Texte peut contenir de 0 à 2 Go de texte.
• Il n'y a aucune différence entre une variable alphanumérique et une variable texte.

Mode non Unicode (compatibilité)

Chaque caractère peut être l'un des 256 caractères ASCII pris en charge par Windows et Mac OS. Pour plus d'informations sur les codes ASCII, reportez-vous à la section Codes ASCII.
• Un champ alphanumérique peut contenir de 0 à 255 caractères (la limite est fixée lors de la définition du champ).
• Une variable de longueur fixe peut contenir de 0 à 255 caractères (la limite est fixée lors de la déclaration de la variable).
• Un champ, une variable ou une expression de type Texte peut contenir de 0 à 32 000 caractères.

Quel que soit le mode, vous pouvez assigner un alpha à un texte et vice-versa, 4D effectue automatiquement la conversion, en tronquant les valeurs si nécessaire. Vous pouvez mélanger du texte et de l'alphanumérique dans les expressions.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Alpha et Texte dans les descriptions des commandes sont indifféremment appelés Chaîne, sauf spécification explicite.

Numérique

Numérique est un terme générique utilisé pour :
• les champs, variables ou expression de type Réel (aussi appelé type Numérique)
• les champs, variables ou expression de type Entier
• les champs, variables ou expression de type Entier long

Les nombres de type Réel sont compris dans l'intervalle ±1.7e±308 (15 chiffres).
Les nombres de type Entier (2 octets) sont compris dans l'intervalle -32 768..32 767
Les nombres de type Entier long (4 octets) sont compris dans l'intervalle -2^31..(2^31)-1

Vous pouvez assigner tout nombre d'un type numérique à un nombre d'un autre type numérique, 4D effectue automatiquement la conversion, en tronquant ou en arrondissant les valeurs si nécessaire. Notez cependant que lorsqu'une valeur est située en-dehors de l'intervalle du type de destination, 4D ne pourra la convertir. Vous pouvez mélanger tous les types de numériques au sein d'une même expression.

Note : Dans ce manuel de référence du langage 4D, quel que soit le type précis des données, les paramètres de type Réel, Entier et Entier long dans les descriptions des commandes sont appelés Numériques, sauf spécification explicite.

Date

• Les variables, champs ou expressions de type Date peuvent être compris entre 1/1/100 et 31/12/32767.
• Une date est structurée sous la forme jour/mois/année (sur un système français).
• Si l'année est fournie avec deux chiffres seulement, 4D déduit que le siècle est le 20e si la valeur est supérieure ou égale à 30 et le 21e si elle est inférieure (ce comportement par défaut peut être modifié à l'aide de la commande SIECLE PAR DEFAUT).

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Date dans les descriptions des commandes sont appelés Date, sauf spécification explicite.

Heure

• Les variables, champs ou expressions de type Heure peuvent être compris entre 00:00:00 et 596 000:00:00.
• Une heure est structurée sous la forme heure:minutes:secondes (avec une version française de 4D).
• Les heures sont stockées dans un format de 24 heures.
• Une valeur de type Heure peut être traitée en tant que nombre. Le nombre correspondant est le nombre de secondes que cette valeur représente. Pour plus d'informations, reportez-vous à la section Opérateurs sur les heures.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Heure dans les descriptions des commandes sont appelés Heure, sauf spécification explicite.

Booléen

Les variables, champs ou expressions de type Booléen peuvent être soit à VRAI soit à FAUX.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Booléen dans les descriptions des commandes sont appelés Booléen, sauf spécification explicite.

Image

Les variables, champs ou expressions de type Image peuvent contenir des images Windows ou Macintosh. En général, ce type accepte toute image pouvant être collée dans le Presse-papiers ou bien lue depuis le disque à l'aide des commandes de 4D ou d'un plug-in.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Image dans les descriptions des commandes sont appelés Image, sauf spécification explicite.

Pointeur

Les variables ou expressions de type Pointeur sont des références à d'autres variables (y compris des tableaux et des éléments de tableaux), à des tables ou à des champs. Il n'existe pas de champs de type Pointeur.

Pour plus d'informations sur les pointeurs, reportez-vous à la section Pointeurs.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Pointeur dans les descriptions des commandes sont appelés Pointeur, sauf spécification explicite.

BLOB

Les champs ou variables de type BLOB sont des séries d'octets (d'une longueur de 0 à 2 Go) que vous pouvez adresser individuellement ou à l'aide des Commandes du thème BLOB. Il n'existe pas d'expressions de type BLOB.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type BLOB dans les descriptions des commandes sont appelés BLOB.

Tableau

Les tableaux ne sont pas véritablement un type de données. Sous cette appellation sont regroupés les différents types de tableaux (comme les tableaux entier, tableaux texte, etc.). Les tableaux sont des variables. Il n'existe pas de champs ni d'expressions de type Tableau. Pour plus d'informations sur les tableaux, reportez-vous à la section Présentation des tableaux.

Note : Dans ce manuel de référence du langage 4D, les paramètres de type Tableau dans les descriptions des commandes sont appelés Tableau, sauf spécification explicite (par exemple Tableau Alpha, Tableau Réel...).

Indéfini

Indéfini n'est pas véritablement un type de données. Une variable dite "indéfinie" est une variable n'ayant pas encore été définie. Une fonction utilisateur (c'est-à-dire une méthode projet qui retourne une valeur) peut retourner une valeur indéfinie si, à l'intérieur de la méthode, le résultat de la fonction ($0) est assigné à une expression indéfinie (une expression issue d'un calcul effectué avec au moins une variable indéfinie). Un champ ne peut pas être indéfini.

Convertir les types de données

Le langage de 4D comporte des fonctions et des opérateurs vous permettant de convertir des types de données en d'autres types, dans la mesure où de telles conversions ont un sens. 4D assure la vérification des types de données. Ainsi, vous ne pouvez pas écrire : "abc"+0.5+!25/12/96!-?00:30:45?, car cette opération génère une erreur de syntaxe.

Le tableau ci-dessous liste les types de données pouvant être convertis, le type dans lequel ils peuvent être convertis, ainsi que les fonctions 4D à utiliser.

Types à convertir	**en Chaîne**	**en Numérique**	**en Date**	**en Heure**
Chaîne		Num	Date	Heure
Numérique (*)	Chaine			
Date	Chaine			
Heure	Chaine			
Booléen		Num		

(*) Les valeurs de type Heure peuvent être utilisées en tant que numériques.

Note : Ce tableau ne traite pas les conversions de données plus complexes obtenues à l'aide d'une combinaison d'opérateurs et d'autres commandes.

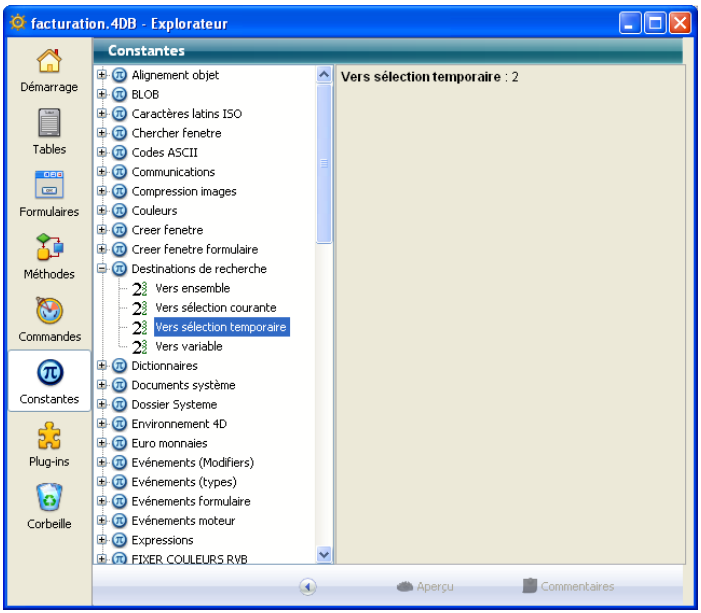
Référence

Conditions et boucles, Constantes, Méthodes, Nommer les objets du langage 4D, Opérateurs, Pointeurs, Présentation des tableaux, Type, Variables.

Une constante est une expression dont la valeur est fixe. Il existe deux types de constantes : les **constantes prédéfinies** que vous pouvez appeler en inscrivant leur nom et les **constantes littérales**, pour lesquelles vous devez saisir une valeur.

Constantes prédéfinies

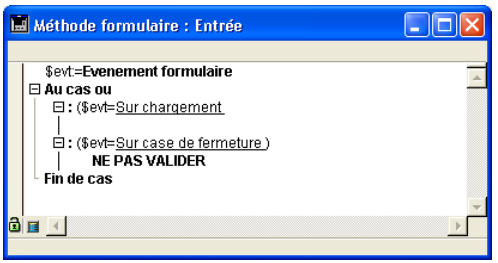
4D propose un ensemble de **constantes prédéfinies**. Ces constantes sont regroupées par thèmes dans la fenêtre de l'Explorateur :



Pour utiliser une constante prédéfinie dans la fenêtre de l'éditeur de méthodes, vous pouvez :
• soit glisser-déposer la constante depuis la fenêtre de l'Explorateur vers la fenêtre de l'éditeur de méthodes,
• soit saisir directement son nom dans la fenêtre de l'éditeur de méthodes. La fonction de saisie prédictive propose les constantes correspondant au contexte de programmation.

Les noms des constantes prédéfinies peuvent contenir jusqu'à 31 caractères.

Les constantes prédéfinies apparaissent soulignées par défaut dans la fenêtre de l'éditeur de méthodes et dans la fenêtre du débogueur :



Dans la fenêtre ci-dessus, Sur chargement est par exemple une constante prédéfinie.

Constantes littérales

4D accepte quatre types de données pour les constantes littérales :

- Chaîne,
- Numérique,
- Date,
- Heure.

Constantes chaînes

Une constante de type chaîne est incluse entre des guillemets droits ("..."). Voici quelques exemples de constantes chaîne :

"Ajouter Enregistrements"
"Aucun enregistrement trouvé."
"Facture"

Une chaîne vide est spécifiée par la succession de deux guillemets ("").

Constantes numériques

Une constante numérique s'écrit comme un nombre réel. Voici quelques exemples de constantes numériques :

27
123,76
0,0076

Les nombres négatifs s'écrivent précédés du signe moins (-). Par exemple:

-27
-123,76
-0,0076

Constantes dates
Une constante de type date est incluse entre deux points d'exclamation (!…!). Dans un environnement français, une date est structurée sous la forme jour/mois/année, une barre oblique "/" séparant les valeurs. Voici quelques exemples de constantes dates :

!1/1/76!
!4/4/04!
!25/12/99!

Une date nulle s'écrit !00/00/00!

Astuce : L'éditeur de méthodes dispose d'un raccourci pour entrer une date nulle. Pour cela, tapez un point d'exclamation (!) et appuyez sur la touche **Entrée**.

Note : Lorsqu'une année est saisie sur deux chiffres, 4D considère qu'elle appartient au 20e siècle, sauf si ce fonctionnement par défaut a été modifié à l'aide de la commande SIECLE PAR DEFAUT.

Constantes heures
Une constante heure est incluse entre deux points d'interrogation (?...?).

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. Sous Mac OS, vous pouvez également utiliser le symbole † (Option+t sur un clavier français).

Avec une version française de 4D, une heure est structurée sous la forme heure:minute:seconde, deux points (:) séparant les valeurs. Les heures sont stockées dans un format de 24 heures.

Voici quelques exemples de constantes heures :

?00:00:00? ` minuit
?09:30:00? ` 9:30 du matin
?13:01:59? ` 13 heures, 1 minute et 59 secondes

Une heure nulle s’écrit ?00:00:00?.

Astuce : L'éditeur de méthodes dispose d'un raccourci pour saisir une heure nulle. Pour cela, tapez un point d'interrogation (?) et appuyez sur la touche Entrée.

Référence

Conditions et boucles, Méthodes, Nommer les objets du langage 4D, Opérateurs, Pointeurs, Types de données, Variables.

Fondamentalement, dans 4D, les données peuvent être stockées de deux manières. Les **champs** stockent les données sur disque, de manière permanente ; les **variables** stockent les données en mémoire, de manière temporaire.

Lorsque vous définissez votre base, vous indiquez à 4D les noms et les types de champs que vous voulez utiliser. C'est pratiquement la même chose pour les variables — vous leur donnez un nom et un type.

Les types de variables suivants correspondent à chacun des types de données :
• Alpha(*) ou Texte : chaîne alphanumérique pouvant contenir jusqu'à 2 Go de texte
• Entier : nombre entier compris entre -32768 et 32767
• Entier long : nombre entier compris entre -2^31 et (2^31) -1
• Réel (ou Numérique) : nombre réel compris entre ±1.7e±308 (15 chiffres)
• Date : date comprise entre 1/1/100 et 31/12/32767
• Heure : heure comprise entre 00:00:00 et 596000:00:00 (secondes depuis minuit)
• Booléen : Vrai ou Faux
• Image : toute image Windows ou Mac OS
• BLOB (Binary Large OBject) : suite d'octets pouvant aller jusqu'à 2 Go
• Pointeur : pointeur vers une table, un champ, une variable, un tableau ou un élément de tableau.

(*) En mode Unicode, les types de variables Alpha et Texte sont identiques. En mode non Unicode (mode compatibilité), un Alpha est une chaîne alphanumérique fixe pouvant comprendre jusqu'à 255 caractères.

Vous pouvez afficher des variables à l'écran (à l'exception des pointeurs et des BLOB), les utiliser pour saisir des données, et les imprimer dans des états. Dans ces cas, elles se comportent exactement comme des champs, et les mêmes contrôles intégrés sont disponibles lorsque vous les créez :

• Formats d'affichage,
• Contrôles de saisie tels que filtres de saisie et valeurs par défaut,
• Filtrages des caractères,
• Enumérations (listes hiérarchiques)
• Valeurs saisissables ou non-saisissables

Les variables peuvent également servir à :

- contrôler des boutons (boutons, cases à cocher, boutons radio, boutons 3D, etc.),
- contrôler des thermomètres, règles et cadrans,
- contrôler des zones de défilement, des pop up menus et des listes déroulantes,
- contrôler des listes hiérarchiques et des menus hiérarchiques,
- contrôler des grilles de boutons, onglets, boutons image, etc.
- afficher les résultats de calculs ne devant pas être sauvegardés.

Créer des variables

Vous pouvez créer des variables simplement en les utilisant ; il n'est pas obligatoire de les déclarer formellement comme vous le faites avec les champs. Par exemple, si vous voulez créer une variable qui contient la date du jour plus 30 jours, il vous suffit d'écrire dans 4D :

```
MaDate:=Date du jour+30
```

MaDate est alors créée et contient la valeur que vous voulez. Le programme interprète la ligne comme "MaDate prend la valeur de la date courante plus 30 jours". Vous pourrez utiliser MaDate à chaque fois que vous le souhaiterez dans votre base. Par exemple, vous pouvez la stocker dans un champ du même type :

```
[MaTable]MonChamp:=MaDate
```

Toutefois, il est généralement conseillé de définir explicitement le type d'une variable. Pour plus d'informations sur le typage des variables dans une base, reportez-vous à la section Commandes du thème compilateur.

Assigner des valeurs aux variables

Vous pouvez donner des valeurs aux variables et/ou récupérer leur valeur. Donner une valeur à une variable s'appelle **assigner une valeur** (ou **affecter une valeur**) et s'effectue à l'aide de l'opérateur d'assignation (:=). L'opérateur d'assignation est également utilisé pour assigner des valeurs aux champs.

L'opérateur d'assignation est le premier moyen pour créer une variable et lui donner une valeur. Vous placez le nom de la variable que vous voulez créer à gauche de l'opérateur. Par exemple :

```
MonNombre:=3
```

crée la variable MonNombre et lui donne la valeur numérique 3. Si MonNombre existait déjà, elle prend simplement la valeur 3.

Bien entendu, les variables ne seraient pas très utiles si vous ne pouviez pas récupérer les valeurs qu'elles contiennent. De nouveau, vous utilisez l'opérateur d'assignation. Si vous devez placer la valeur de MonNombre dans un champ nommé [Produits]Taille, il vous suffit de placer MonNombre à droite de l'opérateur d'assignation :

```
[Produits]Taille:=MonNombre
```

Dans ce cas, [Produits]Taille vaudrait 3. Cet exemple est plutôt simple, mais il illustre le moyen élémentaire dont vous disposez pour transférer des données d'un objet vers un autre en utilisant le langage.

Important : Ne confondez pas l'opérateur d'assignation (:=) avec le signe égal (=) qui est un opérateur de comparaison. L'assignation et la comparaison sont deux opérations très différentes. Pour plus d'informations sur les opérateurs de comparaison, reportez-vous à la section Opérateurs.

Variables locales, process et interprocess

Vous pouvez créer trois types de variables : des variables **locales**, des variables **process** et des variables **interprocess**. La différence entre ces trois types de variables est leur portée, ou les objets pour lesquels elles sont disponibles.

Variables locales

Le premier type de variable est la variable locale. Une variable locale, comme son nom l'indique, est locale à une méthode — c'est-à-dire accessible uniquement à l'intérieur de la méthode dans laquelle elle a été créée et inaccessible à l'extérieur de cette méthode. Pour une variable, être locale à une méthode signifie avoir une portée locale.

Vous utilisez une variable locale lorsque vous souhaitez limiter son fonctionnement à la méthode, pour une des raisons suivantes :
- Eviter des conflits de noms avec les autres variables.
- Utiliser temporairement des valeurs,
- Réduire le nombre de variables process.

Le nom d'une variable locale commence toujours par le signe dollar ($) et peut contenir jusqu'à 31 autres caractères. Si vous saisissez un nom plus long, 4D le tronque pour le ramener à 31 caractères.

Lorsque vous développez une base comportant de nombreuses méthodes et variables, il arrive souvent que vous n'ayez besoin d'utiliser une variable que dans une méthode. Vous pouvez alors créer et utiliser une variable locale, sans devoir vous soucier de l'existence d'une autre variable du même nom ailleurs dans la base.

Fréquemment, dans une base de données, des informations ponctuelles sont demandées à l'utilisateur. La commande Demander peut être appelée pour obtenir ces informations. Elle affiche une boîte de dialogue comportant un message demandant à l'utilisateur de répondre et, lorsque la réponse est validée, la retourne. Généralement, il n'est pas nécessaire de conserver cette information très longtemps dans vos méthodes. C'est l'endroit parfait pour utiliser une variable locale. Voici un exemple :

```
$vsID:=Demander("Saisissez votre numéro d'identification :")
Si (OK=1)
    CHERCHER ([Personnes];[Personnes]ID =$vsID)
Fin de si
```

Cette méthode demande simplement à l'utilisateur de saisir un numéro d'identification. La réponse est placée dans une variable locale, $vsID, puis la méthode la recherche parmi les champs [Personnes]ID. Une fois la méthode terminée, la variable locale $vsID est effacée de la mémoire. Ce fonctionnement est bien adapté puisque la variable n'est utile qu'une seule fois et dans cette méthode uniquement.

Variables process

Le second type de variable est la variable process. Une variable process est "visible" uniquement dans le process où elle a été créée. Elle est utilisable par toutes les méthodes du process et toutes les méthodes appelées depuis le process.

Le nom d'une variable process ne comporte aucun préfixe. Une variable process peut comporter jusqu'à 31 caractères.

En mode interprété, les variables sont gérées dynamiquement : elles sont créées en mémoire et effacées "à la volée". En mode compilé, tous les process que vous créez (process utilisateurs) partagent la même définition des variables process, mais chaque process dispose de sa propre instance pour chaque variable. Par exemple, la variable maVar est une certaine variable dans le process P_1 et une autre variable dans le process P_2.

Un process peut lire et écrire des variables process dans un autre process à l'aide des commandes LIRE VARIABLE PROCESS et ECRIRE VARIABLE PROCESS. Nous vous recommandons de n'utiliser ces commandes que dans le cadre des besoins décrits ci-dessous (qui sont les raisons pour lesquelles ces commandes ont été créées dans 4D) :
• Communication interprocess à des endroits particuliers de votre code
• Gestion du glisser-déposer interprocess
• En client/serveur, communication entre les process sur les postes clients et les procédures stockées exécutées sur le serveur.

Pour plus d'informations, reportez-vous à la section Introduction aux process et à la description de ces commandes.

Variables interprocess

Le troisième type de variable est la variable interprocess. Les variables interprocess sont visibles dans toute la base et sont disponibles pour tous les process.

Le nom d'une variable interprocess débute toujours par le symbole (<>) — formé du symbole "inférieur à" suivi du symbole "supérieur à" — et peut comporter jusqu'à 31 caractères supplémentaires.

Note : Cette syntaxe peut être utilisée sur les plates-formes Windows et Macintosh. En outre, sous Mac OS uniquement, vous pouvez utiliser le symbole "diamant" (Option+v sur un clavier français).

Les variables interprocess sont principalement utilisées pour le partage d'informations entre les process.

En mode client/serveur, chaque poste (client et serveur) partage la même définition des variables interprocess, mais chacun utilise une instance différente d'une variable.

Variables objets dans les formulaires

Dans l'éditeur de formulaires, le fait de donner un nom à un objet actif — bouton, bouton radio, case à cocher, zone de défilement, thermomètre, etc. — crée automatiquement une variable du même nom. Par exemple, si vous créez un bouton appelé MonBouton, une variable MonBouton est également créée. Notez bien que ce nom de variable n'est pas le libellé du bouton, mais son nom.

Ces variables vous permettent de contrôler et de gérer vos objets. Par exemple, lorsque l'utilisateur clique sur un bouton, la variable du bouton prend la valeur 1 ; sinon, le reste du temps, elle est à 0. La variable associée à un thermomètre ou un cadran vous permet de lire et de modifier les valeurs représentées. Par exemple, si l'utilisateur clique dans un thermomètre pour fixer une nouvelle valeur, la valeur de la variable est modifiée en conséquence. De même, si une méthode change la valeur de la variable, le thermomètre est redessiné pour afficher cette nouvelle valeur.

Pour plus d'informations sur les variables et les formulaires, reportez-vous au manuel *Mode Développement* de 4D ainsi qu'à la description de la fonction Evenement formulaire.

Variables système

4D exploite un certain nombre de variables particulières appelées **variables système**. Ces variables vous permettent de contrôler de nombreuses opérations. Toutes les variables système sont des variables process, disponibles à l'intérieur d'un seul process.

La plus importante de toutes est la variable système **OK**. Comme son nom le laisse supposer, elle vous indique si tout est OK dans un process particulier. Est-ce que l'enregistrement a bien été sauvegardé ? Est-ce que l'import d'enregistrements s'est bien déroulé ? Est-ce que l'utilisateur a cliqué sur le bouton OK ? La variable système OK prend la valeur 1 lorsqu'une opération s'est correctement déroulée, et 0 dans le cas contraire.

Pour plus d'informations sur les variables système, reportez-vous à la section traitant des Variables système.

Référence

Conditions et boucles, Constantes, Méthodes, Nommer les objets du langage 4D, Opérateurs, Pointeurs, Présentation des tableaux, Types de données.

4D gère un certain nombre de variables appelées **variables système**. Ces variables vous permettent de contrôler le déroulement de diverses opérations. Les variables système sont toutes des variables process, accessibles uniquement à l'intérieur d'un process. Cette section décrit les variables système de 4D.

OK
La variable système OK est la plus couramment utilisée. En général, elle prend la valeur 1 lorsqu'une opération s'est correctement déroulée, et 0 lorsque l'opération a échoué.
De nombreuses commandes 4D modifient la valeur de la variable système OK. Reportez-vous à la description de chaque commande pour savoir si elle met à jour cette variable système.

Document
La variable système Document contient le chemin d'accès et le nom du dernier fichier disque ayant été ouvert ou créé à l'aide d'une des commandes suivantes :

Ajouter a document
Creer document
ECRIRE FICHIER IMAGE
EXPORTER TEXTE
ECRITURE SYLK
EXPORTER DONNEES
GENERER APPLICATION
IMPRIMER ETIQUETTES
LECTURE DIF
LIRE FICHIER IMAGE
LIRE VARIABLES
Ouvrir fichier ressources
STOCKER ENSEMBLE
UTILISER FILTRE
CHARGER ENSEMBLE
Creer fichier ressources
ECRIRE VARIABLES
ECRITURE DIF
QR ETAT
FIXER FICHIER HISTORIQUE
IMPORTER DONNEES
IMPORTER TEXTE
LECTURE SYLK
LIRE ICONE DOCUMENT
Ouvrir document
Selectionner document
REGLER SERIE

FldDelimit
La variable système FldDelimit contient le code du caractère à utiliser comme délimiteur de champs lorsque vous importez ou exportez du texte. Par défaut, cette valeur est 9, c'est-à-dire le code du caractère Tabulation. Modifiez cette valeur pour changer de délimiteur de champs.

RecDelimit
La variable système RecDelimit contient le code du caractère à utiliser comme délimiteur d'enregistrements lorsque vous importez ou exportez du texte. Par défaut, cette valeur est 13, c'est-à-dire le code du caractère Retour chariot. Modifiez cette valeur pour changer de délimiteur d'enregistrements.

Error
La variable système Error n'est utilisable que dans une méthode installée par la commande APPELER SUR ERREUR. Cette variable contient le code de l'erreur. Les codes des erreurs de 4D et des erreurs Système sont listés dans les sections du thème “Codes d'erreurs”.

MouseDown, MouseX, MouseY, KeyCode, Modifiers et MouseProc
Ces variables système ne sont utilisables que dans une méthode installée par APPELER SUR EVENEMENT.
• La variable système MouseDown prend la valeur 1 si le bouton de la souris a été enfoncé. Sinon, elle prend la valeur 0.
• Si l'événement est un MouseDown (MouseDown=1), les variables système MouseX et MouseY contiennent les coordonnées verticale et horizontale de l'endroit où le clic a eu lieu. Les deux valeurs sont exprimées en pixels et avec le système de coordonnées locales de la fenêtre.
Note : Dans le contexte d'un clic sur un champ image ou une variable image, les variables système MouseX et MouseY retournent les coordonnées locales du clic dans les événements formulaire Sur clic, Sur double clic ainsi que Sur début survol et Sur survol. Pour plus d'informations, reportez-vous à la section Introduction aux images et à la commande SVG Chercher ID element par coordonnees.
• La variable système KeyCode contient le code de la touche ayant été enfoncée. Si la touche enfoncée était une touche de fonction, KeyCode contient un code spécial. Les codes de caractères et les codes des touches de fonction sont listés dans les sections Codes Unicode, Codes ASCII et Codes des touches de fonction.
• La variable système Modifiers contient les codes des modifiers du clavier (**Ctrl/Commande**, **Alt/Option**, **Maj**, **Verr. Maj**). Cette variable n'est significative que dans une méthode d'interruption sur événement installée par la commande APPELER SUR EVENEMENT.
• La variable système MouseProc contient le numéro du process dans lequel le dernier événement a eu lieu.

Référence

Présentation des ensembles, Variables.

Les pointeurs sont des outils de programmation avancée.

Lorsque vous utilisez le langage de 4D, vous vous référez aux différents objets par l'intermédiaire de leur nom — en particulier les tables, champs, variables et tableaux. Pour appeler l'un d'entre eux, vous écrivez simplement son nom. Cependant, il est parfois utile de pouvoir appeler ou référencer ces éléments sans nécessairement connaître leur nom. C'est ce que permettent les pointeurs.

Le concept de pointeur n'est pas tellement éloigné de la vie courante. Vous vous référez souvent à des choses sans connaître leur identité exacte. Par exemple, vous dites à un ami "Allons-y avec ta voiture" au lieu de "Allons-y avec la voiture immatriculée 123 ABD 99". Dans ce cas, vous faites référence à la voiture immatriculée 123 ABD 99 en utilisant l'expression "ta voiture". Par analogie, l'expression "la voiture immatriculée 123 ABD 99" est le nom d'un objet, et "ta voiture" est un pointeur référençant (ou pointant vers) l'objet.

La capacité de se référer à quelque chose sans connaître son identité exacte est très utile. Si votre ami s'achetait une nouvelle voiture, l'expression "ta voiture" serait toujours exacte — ce serait toujours une voiture et vous pourriez toujours aller quelque part avec. Les pointeurs fonctionnent de la même manière. Par exemple, un pointeur peut pointer à un moment donné vers un champ numérique appelé Age, et plus tard vers une variable numérique appelée Ancien âge. Dans les deux cas, le pointeur référence des données numériques pouvant être utilisée dans des calculs. Vous pouvez utiliser des pointeurs pour référencer des tables, des champs, des variables, des tableaux et des éléments de tableaux.

Le tableau suivant vous fournit un exemple de chaque type :

Objet	Référencement	Utilisation	Affectation
Table	vpTble:=->[Table]	TABLE DEFAUT(vpTble->)	n/a
Champ	vpChp:=->[Table]Chp	ALERTE(vpChp->)	vpChp->:="Jean"
Variable	vpVar:=->Variable	ALERTE(vpVar->)	vpVar->:="Jean"
Tableau	vpT:=->Tableau	TRIER TABLEAU(vpT->;>)	COPIER TABLEAU(Tab;vpT->)
Elém. tabl.	vpElem:=->Tableau{1}	ALERTE(vpElem->)	vpElem->:="Jean"

Utiliser des pointeurs : un exemple

Il est plus facile d'expliquer l'utilisation des pointeurs au travers d'un exemple. Cet exemple vous montre comment accéder à une variable par l'intermédiaire d'un pointeur. Nous commençons par créer la variable :

```
MaVar:="Bonjour"
```

MaVar est désormais une variable contenant la chaîne "Bonjour". Nous pouvons alors créer un pointeur vers MaVar :

```
MonPointeur:=->MaVar
```

Le symbole -> signifie "pointer vers" (ce symbole est formé du caractère "tiret" (-) suivi du caractère "supérieur à"). Dans ce cas, il crée un pointeur qui référence ou "pointe vers" MaVar. Ce pointeur est assigné à MonPointeur via l'opérateur d'assignation.

MonPointeur est désormais une variable qui contient un pointeur vers MaVar. MonPointeur ne contient pas "Bonjour", la valeur de MaVar, mais vous pouvez utiliser MonPointeur pour obtenir la valeur contenue dans MaVar. L'expression suivante retourne la valeur de MaVar :

```
MonPointeur->
```

Dans ce cas, la chaîne "Bonjour" est retournée. Lorsque le symbole -> est placé derrière un pointeur, la valeur de l'objet vers lequel pointe le pointeur est récupérée. On dit alors qu'on **dépointe** le pointeur.

Il est important de comprendre que vous pouvez utiliser un pointeur suivi du symbole -> partout où vous auriez pu utiliser l'objet pointé lui-même. Vous pouvez placer l'expression MonPointeur-> partout où vous pourriez utiliser la variable originale MaVar.

Par exemple, l'instruction suivante affiche une boîte de dialogue d'alerte comportant le mot Bonjour :

```
ALERTE(MonPointeur->)
```

Vous pouvez également utiliser MonPointeur pour modifier la valeur de MaVar. Par exemple, l'instruction suivante stocke la chaîne "Au revoir" dans la variable MaVar :

```
MonPointeur->:="Au revoir"
```

Si vous examinez les deux utilisations de l'expression MonPointeur-> ci-dessus, vous constatez que cette expression se comporte exactement comme si vous aviez utilisé MaVar à sa place. En résumé : les deux lignes suivantes effectuent la même opération — elles affichent une boîte de dialogue d'alerte contenant la valeur courante de la variable MaVar :

```
ALERTE(MonPointeur->)
ALERTE(MaVar)
```

Les deux lignes suivantes effectuent la même opération ; elles assignent la chaîne "Au revoir" à MaVar :

```
MonPointeur->:="Au revoir"
MaVar:="Au revoir"
```

Utiliser des pointeurs vers des boutons

Ce paragraphe décrit l'utilisation d'un pointeur pour référencer un bouton. Un bouton (du point de vue du langage) n'est rien d'autre qu'une variable. Bien que les exemples de ce paragraphe utilisent des pointeurs pour référencer des boutons, les concepts présentés s'appliquent à l'utilisation de tout type d'objet pouvant être référencé par un pointeur.

Imaginons que vous disposiez dans vos formulaires d'un certain nombre de boutons devant être actifs ou inactifs. Chaque bouton est associé à une condition qui peut être VRAI ou FAUX. La condition indique s'il faut désactiver ou activer le bouton. Vous pouvez utiliser un test comme celui-ci chaque fois que vous devez désactiver ou activer le bouton :

```
Si (Condition)   ` Si la condition est VRAIE…
   ACTIVER BOUTON (MonBouton)   ` activer le bouton
Sinon         ` Dans l'autre cas…
   INACTIVER BOUTON (MonBouton)   ` désactiver le bouton
Fin de si
```

Vous pouvez avoir besoin, pour chaque bouton que vous avez créé, d'utiliser un test similaire, dans lequel seul le nom du bouton change. Afin d'être plus efficace, vous pouvez créer un pointeur pour référencer chaque bouton puis utiliser une sous-routine pour le test lui-même.

Vous devez utiliser des pointeurs si vous appelez une sous-routine, car il n'est pas possible de faire référence autrement aux variables des boutons. Par exemple, voici une méthode projet nommée FIXER_BOUTON, qui référence un bouton avec un pointeur :

```
` Méthode projet FIXER_BOUTON
` FIXER_BOUTON ( Pointeur ; Booléen )
` FIXER_BOUTON ( -> Bouton ; Activé ou Désactivé )
`
` $1 – Pointeur vers un bouton
` $2 – Booléen. Si VRAI, active le bouton. Si FAUX, désactive le bouton

Si ($2) ` Si la condition est VRAIE…
   ACTIVER BOUTON($1->) ` activer le bouton
Sinon ` Si ce n'est pas le cas…
   INACTIVER BOUTON ($1->) ` désactiver le bouton
Fin de si
```

Vous pouvez appeler la méthode projet FIXER_BOUTON de la manière suivante :

```
   ` ...
FIXER_BOUTON (->bValider;Vrai)
   ` ...
FIXER_BOUTON (->bValider;Faux)
   ` ...
FIXER_BOUTON (->bValider;([Employés]Nom#"")
   ` ...
Boucle ($vlRadioBouton;1;20)
   $vpRadioBouton:=Pointeur vers("r"+Chaine($vlRadioBouton))
   FIXER_BOUTON ($vpRadioBouton;Faux)
Fin de boucle
```

Utiliser des pointeurs vers des tables

Partout où le langage requiert un nom de table, vous pouvez utiliser un pointeur dépointé vers une table.
Pour créer un pointeur vers une table, écrivez une instruction du type :

```
TablePtr:=->[touteTable]
```

Vous pouvez également récupérer un pointeur vers une table à l'aide de la fonction Table. Par exemple :

```
TablePtr:=Table(20)
```

Vous pouvez utiliser le pointeur dépointé dans vos commandes, comme ceci :

```
TABLE PAR DEFAUT(TablePtr->)
```

Utiliser des pointeurs vers des champs

Partout où le langage requiert un nom de champ, vous pouvez utiliser un pointeur dépointé vers un champ.
Pour créer un pointeur vers un champ, écrivez une ligne d'instruction du type :

```
ChampPtr:=->[uneTable]CeChamp
```

Vous pouvez également récupérer un pointeur vers un champ à l'aide de la fonction Champ. Par exemple :

```
ChampPtr:=Champ(1; 2)
```

Vous pouvez utiliser le pointeur dépointé avec les commandes, comme ceci :

```
CHANGER JEU DE CARACTERES(ChampPtr->; "Arial")
```

Utiliser des pointeurs vers des variables

L'exemple fourni au début de cette section illustre l'utilisation d'un pointeur vers une variable :

```
MaVar:="Bonjour"  ` Création de la variable MaVar
MonPointeur:=->MaVar
```

Vous pouvez faire pointer des pointeurs vers des variables interprocess, process et, à compter de la version 2004.1 de 4D, vers des variables locales.

Lorsque vous utilisez des pointeurs vers des variables locales ou des variables process, vous devez veiller à ce que la variable pointée soit bien définie au moment de l'utilisation du pointeur. Rappelons que les variables locales sont supprimées à la fin de l'exécution de la méthode qui les a créées et les variables process à la fin du process dans lequel elles ont été créées.

L'appel d'un pointeur vers une variable qui n'existe plus provoque une erreur de syntaxe en mode interprété (variable indéfinie) mais peut générer une erreur plus conséquente en mode compilé.

Note sur les variables locales : Les pointeurs vers des variables locales permettent dans de nombreux cas d'économiser des variables process. Les pointeurs vers des variables locales peuvent être utilisés uniquement à l'intérieur d'un même process.
Dans le débogueur, lorsque vous affichez un pointeur vers une variable locale déclarée dans une autre méthode, le nom de la méthode d'origine est indiquée entre parenthèses, derrière le pointeur. Par exemple, si vous écrivez dans Méthode1 :

```
$MaVar:="Bonjour"
Méthode2(->$MaVar)
```

Dans Méthode2, le débogueur affichera $1 de la façon suivante :

```
$1                              ->$MaVar (Méthode1)
```

La valeur de $1 sera :

```
$MaVar(Méthode1)                "Bonjour"
```

Utiliser des pointeurs vers des éléments de tableau

Vous pouvez créer un pointeur vers un élément de tableau. Par exemple, les lignes d'instruction suivantes créent un tableau et assignent à une variable appelée ElémPtr un pointeur vers le premier élément :

```
TABLEAU REEL(unTableau; 10) ` Créer un tableau
ElémPtr:=->unTableau{1} ` Créer un pointeur vers l'élément de tableau
```

Vous pouvez alors utiliser le pointeur dépointé pour assigner une valeur à l'élément, comme ceci :

```
ElémPtr->:=8
```

Utiliser des pointeurs vers des tableaux

Vous pouvez créer un pointeur vers un tableau. Par exemple, les lignes d'instruction suivantes créent un tableau et assignent à la variable nommée TabPtr un pointeur vers le tableau :

```
TABLEAU REEL(unTableau; 10) ` Créer un tableau
TabPtr := ->unTableau  ` Créer un pointeur vers le tableau
```

Il est important de comprendre que ce pointeur pointe vers le tableau, et non vers un élément du tableau. Par exemple, vous pourriez utiliser le pointeur dépointé de la manière suivante :

```
TRIER TABLEAU(TabPtr->; >) ` Tri du tableau
```

Si vous devez vous référer au quatrième élément du tableau à l'aide du pointeur, vous pouvez écrire :

```
TabPtr->{4} := 84
```

Utiliser un tableau de pointeurs

Il est souvent utile de disposer d'un tableau de pointeurs référençant un groupe d'objets homogène.

Un exemple d'utilisation typique d'un tel groupe d'objets est une grille de variables dans un formulaire. Chaque variable dans la grille est numérotée de manière séquentielle, par exemple : Var1, Var2,..., Var10. Vous devez souvent vous référer à ces variables, de manière indirecte — par leur numéro. Si vous créez un tableau de pointeurs et faites pointer les pointeurs vers chaque variable, il est alors facile de référencer les variables. Par exemple, pour créer un tableau et initialiser chaque élément, vous pouvez écrire :

```
TABLEAU POINTEUR(tabPointeurs; 10)   ` Créer un tableau de 10 pointeurs
Boucle ($i; 1; 10)   ` Boucle une fois par variable
   tabPointeurs{$i}:=Pointeur vers("Var"+Chaine($i))
      ` Initialiser chaque élément du tableau
Fin de boucle
```

La fonction Pointeur vers retourne un pointeur vers l'objet passé en paramètre.

Pour référencer une des variables, il suffit d'appeler les éléments du tableau. Par exemple, pour remplir les variables avec les dix prochains jours (en supposant que les variables soient de type Date), vous pourriez écrire :

```
Boucle ($i; 1; 10) ` Boucle une fois par variable
   tabPointeurs{$i}->:=Date du jour+$i ` Assigner les jours
Fin de boucle
```

Sélectionner un bouton avec un pointeur

Si vous disposez d'un groupe homogène de boutons radio dans un formulaire, vous devrez souvent pouvoir définir rapidement leur état. La solution consistant à référencer chacun d'entre eux par son nom n'est pas très pratique. Imaginons que vous disposiez d'un groupe de cinq boutons radio libellés Bouton1, Bouton2,..., Bouton5.

Dans un groupe de boutons radio, seul l'un d'entre eux est sélectionné. Le numéro du bouton sélectionné peut être stocké dans un champ numérique. Par exemple, si le champ [Préférences]EtatBouton contient 3, alors Bouton3 est sélectionné. Dans la méthode formulaire, vous pouvez utiliser le code suivant pour paramétrer les boutons :

```
Au cas ou
    :(Evenement formulaire=Sur chargement)
        ` ...
        Au cas ou
            : ([Préférences]EtatBouton = 1)
                Bouton1:=1
            : ([Préférences]EtatBouton = 2)
                Bouton2:=1
            : ([Préférences]EtatBouton = 3)
                Bouton3:=1
            : ([Préférences]EtatBouton = 4)
                Bouton4:=1
            : ([Préférences]EtatBouton = 5)
                Bouton5:=1
        Fin de cas
        ` ...
Fin de cas
```

Un cas particulier doit être testé pour chaque bouton radio. La méthode peut être très longue si le formulaire contient de nombreux boutons radio. Heureusement, vous pouvez utiliser des pointeurs pour résoudre ce problème.
La fonction Pointeur vers vous permet de retourner un pointeur vers un bouton radio. L'exemple suivant utilise un pointeur de ce type pour référencer le bouton radio devant être sélectionné.

Voici la méthode optimisée :

```
Au cas ou
    :(Evenement formulaire=Sur chargement)
            ` ...
        $vpRadio:=Pointeur vers("Bouton"+Chaine([Préférences]EtatBouton))
        $vpRadio->:=1
            ` ...
Fin de cas
```

Le numéro du bouton radio traité doit être stocké dans le champ [Préférences]EtatBouton. Cette opération peut être effectuée dans la méthode formulaire, pour l'événement formulaire Sur clic :

```
[Préférences]EtatBouton:=Bouton1+(Bouton2*2)+(Bouton3*3)+(Bouton4*4)+(Bouton5*5)
```

Passer des pointeurs aux méthodes

Vous pouvez passer un pointeur en tant que paramètre d'une méthode. A l'intérieur de la méthode, vous pouvez modifier l'objet référencé par le pointeur. Par exemple, la méthode suivante, RECOIT DEUX, reçoit deux paramètres qui sont des pointeurs. Elle passe l'objet référencé par le premier paramètre en caractères majuscules, et l'objet référencé par le second paramètre en caractères minuscules :

```
    ` Méthode projet RECOIT DEUX
    ` $1 – Pointeur vers un champ ou une variable de type Chaîne. Passe la chaîne en majuscules.
    ` $2 – Pointeur vers un champ ou une variable de type Chaîne. Passe la chaîne en minuscules.
$1->:=Majusc($1->)
$2->:=Minusc($2->)
```

L'instruction suivante emploie la méthode RECOIT DEUX pour passer un champ en caractères majuscules et une variable en caractères minuscules :

```
RECOIT DEUX (->[MaTable]MonChamp; ->MaVar)
```

Si le champ, [MaTable]MonChamp, contenait la chaîne "dupont", celle-ci deviendrait "DUPONT". Si la variable MaVar contenait la chaîne "BONJOUR", celle-ci deviendrait "bonjour".

Dans la méthode RECOIT DEUX (et, en fait, à chaque fois que vous utilisez des pointeurs), il est important que les types de données des objets référencés soient corrects. Dans l'exemple précédent, les pointeurs doivent pointer vers des objets contenant une chaîne ou un texte.

Pointeurs vers des pointeurs

Si vous aimez compliquer les choses à l'extrême (bien que cela ne soit pas nécessaire dans 4D), vous pouvez utiliser des pointeurs pour référencer d'autres pointeurs. Examinons l'exemple suivant :

```
MaVar := "Bonjour"
PointeurUn := ->MaVar
PointeurDeux := ->PointeurUn
(PointeurDeux->)-> := "Au revoir"
ALERTE((PointeurDeux->)->)
```

Cet exemple affiche une boîte de dialogue d'alerte contenant "Au revoir".

Voici la description de chaque ligne de l'exemple :

• MaVar := "Bonjour"
→ Cette ligne place simplement la chaîne "Bonjour" dans la variable MaVar.

• PointeurUn := ->MaVar
→ PointeurUn contient désormais un pointeur vers MaVar.

• PointeurDeux :=->PointeurUn
→ PointeurDeux (une nouvelle variable) contient un pointeur vers PointeurUn, qui, elle, pointe vers MaVar.

• (PointeurDeux->)-> := "Au revoir"
→ PointeurDeux-> référence le contenu de PointeurUn, qui elle-même référence MaVar. Par conséquent, (PointeurDeux->)-> référence le contenu de MaVar. Donc, dans ce cas, la valeur "Au revoir" est assignée à la MaVar.

• ALERTE ((PointeurDeux->)->)
→ C'est ici la même chose que précédemment : PointeurDeux-> référence le contenu de PointeurUn, qui elle-même référence MaVar. Par conséquent, (PointeurDeux->)-> référence le contenu de MaVar. Donc, dans ce cas, la boîte de dialogue d'alerte affiche le contenu de maVar.

La ligne suivante place la valeur "Bonjour" dans MaVar :

```
(PointeurDeux->)->:="Bonjour"
```

La ligne suivante récupère "Bonjour" à partir de MaVar et la place dans NouvelleVar :

```
NouvelleVar:=(PointeurDeux->)->
```

Important : Vous devez utiliser des parenthèses lors des déréférencements multiples.

Référence

Conditions et boucles, Constantes, Méthodes, Nommer les objets du langage 4D, Opérateurs, Présentation des tableaux, Tableaux et pointeurs, Types de données, Variables.

Cette section décrit les conventions d'écriture employées pour les nombreux objets du langage de 4D. Le nom de chaque objet doit respecter les règles suivantes :

• Un nom doit commencer par un caractère alphabétique (une lettre).
• Le nom peut ensuite contenir des caractères alphabétiques, des caractères numériques, des espaces et des tirets bas (_).
• Les virgules, barres de fraction, guillemets et deux points (:) sont interdits.
• Les caractères réservés car utilisés comme opérateurs, comme l'astérisque (*) ou le +, sont interdits.
• 4D ignore les espaces superflus.

Note : Des règles supplémentaires sont à respecter lorsque les objets doivent être manipulés via le SQL : seuls les caractères _0123456789abcdefghijklmnopqrstuvwxyz sont acceptés, et le nom ne doit pas comporter de mot-clé SQL (commande, attribut, etc.). La zone "SQL" de l'inspecteur de l'éditeur de Structure signale automatiquement les caractères non autorisés dans un nom de table ou de champ.

Tables

Vous indiquez qu'un objet est une table en plaçant son nom entre crochets : [...]. Un nom de table peut contenir jusqu'à 31 caractères.

Exemples

```
TABLE PAR DEFAUT ([Commandes])
FORMULAIRE ENTREE ([Clients]; "Entrée")
AJOUTER ENREGISTREMENT ([Lettres])
```

Champs

Vous indiquez qu'un objet est un champ en spécifiant d'abord la table à laquelle il appartient. Le nom du champ se place immédiatement derrière celui de la table. Un nom de champ peut contenir jusqu'à 31 caractères.

Ne faites pas commencer un nom de champ par le tiret bas (_). Ce caractère est réservé pour l'utilisation des plug-ins. Lorsque 4D rencontre dans l'éditeur de méthodes un nom de champ débutant par le tiret bas, le caractère est supprimé.

Exemples

```
[Commandes]Total:=Somme([Ligne]Montant)
CHERCHER([Clients];[Clients]Nom="Dupont")
[Lettres]Texte:=Capitaliser texte ([Lettres]Texte)
```

Variables interprocess

Vous indiquez qu'un objet est une variable interprocess en faisant précéder son nom des symboles (<>), formés des caractères "inférieur à" suivi de "supérieur à".

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'une variable interprocess peut contenir jusqu'à 31 caractères, symbole <> non compris.

Exemples

```
<>vlProcessID:=Numero du process courant
<>vsKey:=Caractere(KeyCode)
Si (<>vtNom#"")
```

Variables process

Vous indiquez qu'un objet est une variable process en écrivant simplement son nom (qui ne doit pas commencer par les symboles $ ou <>). Ce nom peut contenir jusqu'à 31 caractères.

Exemples

```
vrGrandTotal:=Somme([Comptes]Montant)
Si (bValider=1)
vsNomCourant:=""
```

Variables locales

Vous indiquez qu'un objet est une variable locale en faisant précéder son nom du symbole dollar ($). Le nom d'une variable locale peut contenir jusqu'à 31 caractères, signe dollar non compris.

Exemples

```
Boucle ($vlEnregistrement; 1; 100)
Si ($vsTempVar="No")
$vsMaChaîne:="Bonjour à tous"
```

Tableaux

Vous indiquez qu'un objet est un tableau en écrivant simplement son nom, qui est celui que vous passez à une commande de déclaration de tableau (par exemple TABLEAU ENTIER LONG) lorsque vous créez le tableau. Les tableaux sont des variables, et comme pour les variables, il existe trois types de tableaux qui se différencient par leur portée :

- Tableaux interprocess,
- Tableaux process,
- Tableaux locaux.

Tableaux interprocess
Le nom d'un tableau interprocess est précédé du symbole (<>), formé des caractères "inférieur à" suivi de "supérieur à".

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'un tableau interprocess peut contenir jusqu'à 31 caractères, symbole <> non compris.

Exemples

```
TABLEAU TEXTE(<>atSujets;Enregistrements dans table([Topics]))
TRIER TABLEAU (<>asMotsClés; >)
TABLEAU ENTIER(<>aiGrosTableau;10000)
```

Tableaux process
Vous indiquez qu'un objet est un tableau process en écrivant simplement son nom (qui ne doit pas commencer par les symboles $ ou <>). Ce nom peut contenir jusqu'à 31 caractères.

Exemples

```
TABLEAU TEXTE(atSujets;Enregistrements dans table([Topics]))
TRIER TABLEAU (asMotsClés; >)
TABLEAU ENTIER(aiGrosTableau;10000)
```

Tableaux locaux
Un tableau est déclaré local lorsque son nom est précédé du signe dollar ($). Le nom d'un tableau local peut contenir jusqu'à 31 caractères, signe dollar non compris.

Exemples

```
TABLEAU TEXTE($atSujets;Enregistrements dans table([Topics]))
TRIER TABLEAU ($asMotsClés; >)
TABLEAU ENTIER($aiGrosTableau;10000)
```

Eléments de tableaux
Vous désignez un élément d'un tableau local, process ou interprocess à l'aide d'accolades ({…}). L'élément référencé (l'indice) est indiqué par une expression numérique.

Exemples

```
   ` Adresser un élément d'un tableau interprocess
Si (<>asMotsClés{1}="Stop")
<>atSujets{$vlElem}:=[Topics]Sujet
$viValeurSuivante:=<>aiGrosTableau{Taille tableau(<>aiGrosTableau)}

   ` Adresser un élément d'un tableau process
Si (asMotsClés{1}="Stop")
atSujets{$vlElem}:=[Topics]Sujet
$viValeurSuivante:=aiGrosTableau{Taille tableau(aiGrosTableau)}

   ` Adresser un élément d'un tableau local
Si ($asMotsClés{1}="Stop")
$atSujets{$vlElem}:=[Topics]Sujet
$viValeurSuivante:=$aiGrosTableau{Taille tableau($aiGrosTableau)}
```

Eléments de tableaux à deux dimensions
Vous désignez un élément d'un tableau à deux dimensions à l'aide d'une double paire d'accolades ({…}). L'élément référencé (l'indice) est indiqué par deux expressions numériques dans deux paires d'accolades.

Exemples

```
   ` Adresser un élément d'un tableau interprocess à deux dimensions
Si (<>asMotsClés{$vlLigneSuivante}{1}="Stop")
<>atSujets{10}{$vlElem}:=[Topics]Sujet
$viValeurSuivante:=<>aiGrosTableau{$vlSet}{Taille tableau(<>aiGrosTableau{$vlSet})}
```

```
    ` Adresser un élément d'un tableau process à deux dimensions
Si (asMotsClés{$vlLigneSuivante}{1}="Stop")
atSujets{10}{$vlElem}:=[Topics]Sujet
$viValeurSuivante:=aiGrosTableau{$vlSet}{Taille tableau(aiGrosTableau{$vlSet})}

    ` Adresser un élément d'un tableau interprocess à deux dimensions
Si ($asMotsClés{$vlLigneSuivante}{1}="Stop")
$atSujets{10}{$vlElem}:=[Topics]Sujet
$viValeurSuivante:=$aiGrosTableau{$vlSet}{Taille tableau($aiGrosTableau{$vlSet})}
```

Formulaires

Vous indiquez qu'un objet est un formulaire en utilisant une expression de type chaîne alphanumérique qui représente son nom. Le nom d'un formulaire peut contenir jusqu'à 31 caractères.

Exemples

```
FORMULAIRE ENTREE([Personnes];"Entrée")
FORMULAIRE SORTIE([Personnes]; "Sortie")
DIALOGUE([Stock];"Boîte de note"+Chaine($vlStage))
```

Méthodes

Vous indiquez qu'un objet est une méthode (sous-routine ou fonction utilisateur) en saisissant son nom. Ce nom peut contenir jusqu'à 31 caractères.

Note : Une méthode qui ne retourne pas de résultat est appelée une procédure. Une méthode qui retourne un résultat est appelée une fonction utilisateur.

Exemples

```
Si (Nouveau client)
EFFACER VALEURS DUPLIQUEES
APPLIQUER A SELECTION ([Employés];AUGMENTER SALARIES)
```

Conseil : Nous vous recommandons d'adopter, pour nommer vos méthodes, la même convention que celle utilisée dans le langage de 4D : écrivez les noms de vos procédures en caractères majuscules, et vos fonctions en minuscules avec la première lettre en majuscule. Ainsi, lorsque vous réouvrirez une base au bout de plusieurs mois, vous identifierez immédiatement si une méthode retourne ou non un résultat, en regardant son nom dans la fenêtre de l'Explorateur.

Note : Lorsque vous souhaitez appeler une méthode, vous saisissez simplement son nom. Toutefois, certaines commandes intégrées telles que APPELER SUR ERREUR, ainsi que les commandes des plug-ins, nécessitent que vous passiez le nom d'une méthode en tant que chaîne lorsqu'un paramètre de type méthode est requis (cf. exemples ci-dessous).

Exemples

```
    ` Cette commande attend une méthode (fonction) ou une formule
CHERCHER PAR FORMULE ([aTable];Recherche Spéciale)
    ` Cette commande attend une méthode (procédure) ou une formule
APPLIQUER A SELECTION ([Employés];AUGMENTER SALARIES)
    ` Mais cette commande attend un nom de méthode
APPELER SUR EVENEMENT ("GERER EVENEMENTS")
    ` Et cette commande de plug-in attend un nom de méthode
WR APPELER SUR ERREUR ("WR GERER ERREURS")
```

Les méthodes peuvent accepter des paramètres (ou arguments). Le paramètres sont passés à la méthode entre parenthèses, à la suite du nom de la méthode. Les paramètres sont séparés par des points virgule (;). Les paramètres sont passés à la méthode appelée en tant que variables locales numérotées séquentiellement : $1, $2,..., $n. De plus, plusieurs paramètres consécutifs (s'ils sont les derniers) peuvent être adressés à l'aide de la syntaxe ${n}où n, expression numérique, est le numéro du paramètre.
A l'intérieur d'une fonction, la variable locale $0 contient la valeur à retourner.

Exemples

```
    ` Dans ELIMINER ESPACES, $1 est pointeur sur le champ [Personnes]Nom
ELIMINER ESPACES (->[Personnes]Nom)

    ` Dans Créateur tableau :
    ` - $1 est un numérique qui vaut 1
    ` - $2 est un numérique qui vaut 5
    ` - $3 est un texte ou un alpha qui vaut "Super"
    ` - La valeur résultante est assignée à $0
$vsRésultat:=Créateur tableau (1; 5; "Super")

    ` Dans Poubelle :
    ` - Les trois paramètres sont de type Texte ou Alpha
    ` - Vous pouvez y accéder par $1, $2 ou $3
    ` - Vous pouvez y accéder en écrivant, par exemple, ${$vlParam} où $vlParam vaut 1, 2
                                                                                ou 3
    ` - La valeur résultante est assignée à $0
vtClone:=Poubelle ("est"; "le"; "il")
```

Commandes de plug-ins (procédures, fonctions et zones externes)

Vous indiquez qu'un objet est une commande de plug-in en écrivant son nom tel qu'il est défini dans le plug-in. Le nom d'une commande de plug-in peut contenir jusqu'à 31 caractères.

Exemples

WR SUPPRIMER SELECTION (wrZone)
$pvNouvelleZone:=***PV Creer zone hors ecran***

Ensembles

Dans 4D, il existe deux types d'ensembles qui se distinguent par leur portée :

- Ensembles interprocess,
- Ensembles process.

On peut également distinguer un troisième type d'ensemble, spécifique à 4D Server:
- Ensembles clients.

Ensembles interprocess
Un ensemble est déclaré interprocess lorsque son nom, qui est une expression de type chaîne alphanumérique, est précédé du symbole (<>), formé des caractères "inférieur à" suivi de "supérieur à".

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'un ensemble interprocess peut comporter jusqu'à 255 caractères, symbole <> non compris.

Ensembles process
Vous déclarez un ensemble process en passant simplement une expression de type chaîne qui représente son nom (et qui ne doit pas débuter par les symboles <> ou $). Le nom d'un ensemble process peut comporter jusqu'à 255 caractères.

Ensembles client
Le nom d'un ensemble client doit être précédé du symbole dollar ($). Ce nom peut comporter jusqu'à 255 caractères, symbole dollar non compris.

Note : Les ensembles sont gérés par le serveur. Dans certains cas, pour des raisons particulières ou d'optimisation, vous pourrez avoir besoin d'utiliser des ensembles localement, sur les postes clients. Pour cela, il vous suffit de créer des ensembles client.

Exemples

```
    ` Ensembles interprocess
UTILISER ENSEMBLE("<>Enregistrements supprimés")
NOMMER ENSEMBLE([Clients];"<>Commandes clients")
Si (Enregistrements dans ensemble("<>Sélection"+Chaine($i))>0)
    ` Ensembles process
UTILISER ENSEMBLE("Enregistrements supprimés")
NOMMER ENSEMBLE([Clients];"Commandes clients")
Si (Enregistrements dans ensemble("Sélection"+Chaine($i))>0)
    ` Ensembles client
UTILISER ENSEMBLE("$Enregistrements supprimés")
NOMMER ENSEMBLE([Clients];"$Commandes clients")
Si (Enregistrements dans ensemble("$Sélection"+Chaine($i))>0)
```

Sélections temporaires

Dans 4D, il existe deux types de sélections temporaires, qui se distinguent par leur portée :

- Sélections temporaires interprocess,
- Sélections temporaires process.

Sélections temporaires interprocess
Vous désignez une sélection temporaire interprocess en utilisant une expression de type chaîne débutant par le symbole (<>), formé des caractères “inférieur à” suivi de “supérieur à”.

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. De plus, sous Mac OS, vous pouvez également utiliser le caractère "diamant" (Option + v sur un clavier français).

Le nom d'une sélection temporaire interprocess peut contenir jusqu’à 255 caractères, symbole <> non compris.

Sélections temporaires process
Vous déclarez une sélection temporaire process en passant simplement une expression de type chaîne qui représente son nom (et qui ne doit pas débuter par les symboles <> ou $). Le nom d’une sélection temporaire process peut comporter jusqu’à 255 caractères.

Exemples

```
    ` Sélection temporaire interprocess
UTILISER SELECTION([Clients];"<>ParCodePostal")
    ` Sélection temporaire process
UTILISER SELECTION([Clients];"ParCodePostal")
```

Process

En mode mono-utilisateur, ou sur le poste client en mode client/serveur, il existe deux types de process :

- Process globaux,
- Process locaux.

Process globaux
Vous déclarez un process global en passant une expression de type chaîne de caractères qui représente son nom (qui ne doit pas commencer par le symbole **$**). Le nom d'un process peut comporter jusqu'à 255 caractères.

Process locaux
Vous déclarez un process local lorsque son nom est précédé du symbole dollar (**$**). Le nom d'un process peut comporter jusqu'à 255 caractères, symbole dollar non compris.

Exemple

```
    ` Lancer le process global "Ajouter Clients"
$vlProcessID:=Nouveau process("P_AJOUT_CLIENTS";48*1024;"Ajouter Clients")
    ` Lancer le process local "$Suivre Souris"
$vlProcessID:=Nouveau process("P_SUIVRE_SOURIS";16*1024;"$Suivre Souris")
```

Résumé des conventions d'écriture dans 4D

Le tableau suivant résume les conventions utilisées par 4D pour nommer les objets dans les méthodes.

Objet	Long. max.	Exemple
Table	31	[Factures]
Champ	31	[Employés]Nom
Variable interprocess	<> + 31	<>vlProcessSuivantID
Variable process	31	vsNomCourant
Variable locale	$ + 31	$vlCompteurLocal
Formulaire	31	"Formulaire Web perso"
Tableau interprocess	<> + 31	<>apTableaux
Tableau process	31	asGenre
Tableau local	$ + 31	$atValeurs
Méthode	31	M_AJOUTER_CLIENTS
Commande de plug-in	31	WR INSERER TEXTE
Ensemble interprocess	<> + 255	"<>Enregistrements à archiver"

Ensemble process	255	"Enregistrements actuels
Ensemble client	$ + 255	"$Sujets précédents"
Sélection temporaire	255	"Employés de A à Z"
Sélection temporaire interprocess	<> + 255	"<>Employés de Z à A"
Process local	$ + 255	"$SuivreEvénements"
Process global	255	"P_MODULE_FACTURES"
Sémaphore	255	"monsémaphore"

Résoudre les conflits de noms

Si un objet d'un certain type a le même nom qu'un autre objet d'un autre type (par exemple, si un champ est baptisé Personnes et qu'une variable est également nommée Personnes), 4D utilise un système de priorités pour identifier l'objet. Veillez à utiliser des noms uniques pour les différents éléments de votre base.

4D identifie les noms utilisés dans les méthodes en fonction de l'ordre de priorité suivant :

1. Champs
2. Commandes
3. Méthodes
4. Routines de plug-ins
5. Constantes prédéfinies
6. Variables

Par exemple, 4D dispose d'une fonction interne appelée Date. Si vous appelez Date une de vos méthodes, 4D considérera "Date" comme étant la fonction interne et non votre méthode. Vous ne pourrez pas appeler votre méthode. En revanche, si vous nommez un champ "Date", 4D considérera que vous souhaitez appeler votre champ et non la fonction intégrée.

Référence
Constantes, Méthodes, Opérateurs, Pointeurs, Présentation des tableaux, Types de données, Variables.

Quelle que soit la simplicité ou la complexité d'une méthode, vous utiliserez toujours un ou plusieurs types de structure de programmation. Les structures de programmation déterminent si et dans quel ordre les lignes d'instructions sont exécutées à l'intérieur d'une méthode. Il existe trois types de structures :

- séquentielle,
- conditionnelle,
- répétitive.

Le langage de 4D dispose d'instructions permettant de contrôler chacune de ces structures.

Structures séquentielles
Une structure séquentielle est une structure simple, linéaire. Une séquence est une série d'instructions que 4D exécute les unes après les autres, de la première à la dernière. Par exemple :

```
FORMULAIRE SORTIE ([Personnes]; "Listing")
TOUT SELECTIONNER ([Personnes])
VISUALISER SELECTION ([Personnes])
```

Une instruction d'une ligne, fréquemment utilisée pour les méthodes objet, est le cas le plus simple de structure séquentielle. Par exemple :

```
[Personnes]Nom:=Majusc([Personnes]Nom)
```

Note : Les mots-clés Debut SQL / Fin SQL permettent de délimiter des structures séquentielles destinées à être exécutées par le moteur SQL de 4D. Pour plus d'informations, reportez-vous à la description de ces mots-clés.

Structures conditionnelles
Une structure conditionnelle permet aux méthodes de tester une condition et d'exécuter des séquences d'instructions différentes en fonction du résultat. La condition est une expression booléenne, c'est-à-dire pouvant retourner VRAI ou FAUX. L'une des structures conditionnelles est la structure Si...Sinon...Fin de si, qui aiguille le déroulement du programme vers une séquence ou une autre. L'autre structure conditionnelle est la structure Au cas ou...Sinon...Fin de cas, qui aiguille le programme vers une séquence parmi une ou plusieurs alternatives.

Structures répétitives (ou "boucles")
Il est très courant, lorsque vous écrivez des méthodes, de rencontrer des cas où vous devez répéter une séquence d'instructions un certain nombre de fois. Pour traiter ces besoins, le langage vous propose trois structures répétitives : Boucle...Fin de boucle, Tant que...Fin tant que, et Repeter...Jusque. Les boucles sont contrôlées de deux manières : soit elles se répètent jusqu'à ce qu'une condition soit remplie, soit elles se répètent un nombre fixé de fois. Chaque structure répétitive peut être utilisée de l'une ou l'autre manière, mais les boucles Tant que et Repeter sont mieux adaptées à la répétition jusqu'à ce qu'une condition soit remplie, alors que les boucles Boucle sont mieux adaptées à la répétition un certain nombre de fois.

Note : 4D vous permet d'imbriquer des structures de programmation (Si/Tant que/Boucle/Au cas ou/Repeter) jusqu'à une "profondeur" de 512 niveaux.

Référence

Méthodes, Opérateurs logiques.

La syntaxe de la structure conditionnelle Si...Sinon...Fin de si est la suivante :

```
Si (Expression_booléenne)
    instruction(s)
Sinon
    instruction(s)
Fin de si
```

A noter que l'élément Sinon est optionnel, vous pouvez écrire :

```
Si (Expression_booléenne)
    instruction(s)
Fin de si
```

La structure Si...Sinon...Fin de si permet à votre méthode de choisir dans une alternative, en fonction du résultat, VRAI ou FAUX, d'un test (une expression booléenne).

Si l'expression booléenne est VRAIE, les instructions qui suivent immédiatement le test sont exécutées. Si l'expression booléenne est FAUSSE, les instructions suivant la ligne Sinon sont exécutées. Le Sinon est optionnel ; lorsqu'il est omis, c'est la première ligne d'instructions suivant le Fin de si (s'il y en a une) qui est exécutée.

Exemple

```
    ` Demander à l'utilisateur de saisir un nom
$Rech:=Demander("Saisissez un nom :")
Si (OK=1)
    CHERCHER([Personnes]; [Personnes]Nom=$Rech)
Sinon
    ALERTE("Vous n'avez pas saisi de nom.")
Fin de si
```

Astuce : Il n'est pas obligatoire que des instructions soient exécutées dans chaque branche de l'alternative. Lorsque vous développez un algorithme, ou lorsque vous poursuivez un but précis, rien ne vous empêche d'écrire :

```
Si (Expression_booléenne)
Sinon
    instruction(s)
Fin de si
```

ou :

```
Si (Expression_booléenne)
    instruction(s)
Sinon
Fin de si
```

Référence

Au cas ou...Sinon...Fin de cas, Boucle...Fin de boucle, Conditions et boucles, Repeter...Jusque, Tant que...Fin tant que.

La syntaxe de la structure conditionnelle Au cas ou...Sinon...Fin de cas est la suivante :

```
Au cas ou
    : (Expression_booléenne)
        instruction(s)
    : (Expression_booléenne)
        instruction(s)
    .
    .
    .

    : (Expression_booléenne)
        instruction(s)
Sinon
    instruction(s)
Fin de cas
```

A noter que l'élément Sinon est optionnel, vous pouvez donc écrire :

```
Au cas ou
    : (Expression_booléenne)
        instruction(s)
    : (Expression_booléenne)
        instruction(s)
    .
    .
    .

    : (Expression_booléenne)
        instruction(s)
Fin de cas
```

Tout comme la structure Si...Sinon...Fin de si, la structure Au cas ou...Sinon...Fin de cas permet également à votre méthode de choisir parmi plusieurs séquences d'instructions. A la différence de la structure précédente, le Au cas ou...Sinon...Fin de cas peut tester un nombre illimité d'expressions booléennes et exécuter la séquence d'instructions correspondant à la valeur VRAI.

Chaque expression booléenne débute par le caractère deux points (:). La combinaison de deux points et d'une expression booléenne est appelé un cas. Par exemple, la ligne suivante est un cas :

```
: (bValider=1)
```

Seules les instructions suivant le premier cas VRAI (et ce, jusqu'au cas suivant) seront exécutées. Si aucun des cas n'est VRAI, aucune instruction n'est exécutée (s'il n'y a pas d'élément Sinon).

Vous pouvez placer une instruction Sinon après le dernier cas. Si tous les cas sont FAUX, les instructions suivant le Sinon seront exécutées.

Exemple
Cet exemple teste une variable numérique et affiche une boîte de dialogue d'alerte comportant un simple mot :

```
Au cas ou
    : (vRésult = 1)   ` Test si le numéro est 1
        ALERTE("Un.")   ` Si c'est 1, afficher une alerte
    : (vRésult = 2)   ` Test si le numéro est 2
        ALERTE("Deux.")   ` Si c'est 2, afficher une alerte
    : (vRésult = 3) ` Test si le numéro est 3
        ALERTE("Trois.")   ` Si c'est 3, afficher une alerte
Sinon   ` Si ce n'est ni 1 ni 2 ni 3, afficher une alerte
    ALERTE("Ce n'est ni un, ni deux, ni trois.")
Fin de cas
```

A titre de comparaison, voici la version avec Si...Sinon...Fin de si de la même méthode :

```
Si (vRésult = 1)    ` Test si le numéro est 1
    ALERTE ("Un.")    ` Si c'est 1, afficher une alerte
Sinon
    Si (vRésult = 2)    ` Test si le numéro est 2
        ALERTE ("Deux.")    ` Si c'est 2, afficher une alerte
    Sinon
```

```
            Si (vRésult = 3)    ` Test si le numéro est 3
                ALERTE ("Trois.")    ` Si c'est 3, afficher une alerte
            Sinon    ` Si ce n'est ni 1, 2 ni 3, afficher l'alerte
                ALERTE ("Ce n'est ni un, ni deux, ni trois.")
            Fin de si
        Fin de si
    Fin de si
```

Rappelez-vous qu'avec une structure de type Au cas ou...Sinon...Fin de cas, seul le premier cas VRAI rencontré est exécuté. Même si d'autres cas sont VRAIS, seules les instructions suivant le premier cas VRAI seront prises en compte.

Par conséquent, lorsque vous testez dans la même méthode des cas simples et des cas complexes, vous devez placer les cas complexes **avant** les cas simples, sinon ils ne seront jamais exécutés. Par exemple, si vous souhaitez traiter le cas simple (vRésult=1) et le cas complexe (vRésult=1) & (vDemande#2) et que vous structurez la méthode de la manière suivante :

```
Au cas ou
    : (vRésult = 1)
        ...  `instruction(s)
    : ((vRésult = 1) & (vDemande#2))   `← Les instructions suivant ce cas ne seront
                                          ` JAMAIS exécutées
        ...  `instruction(s)
Fin de cas
```

... les instructions associées au cas complexe ne seront jamais exécutées. En effet, pour que ce cas soit VRAI, ses deux conditions booléennes doivent l'être. Or, la première condition est celle du cas simple situé précédemment. Lorsqu'elle est VRAIE, le cas simple est exécuté et 4D sort de la structure conditionnelle, sans évaluer le cas complexe. Pour que ce type de méthode fonctionne, vous devez écrire :

```
Au cas ou
    : (vRésult = 1) & (vDemande#2)   `← Les cas complexes doivent toujours être placés
                                        ` en premier
        ...  `Instruction(s)
    : (vRésult = 1)
        ...  `Instruction(s)
Fin de cas
```

Astuce

Il n'est pas obligatoire que des instructions soient exécutées dans toutes les alternatives. Lorsque vous développez un algorithme, ou lorsque vous poursuivez un but particulier, rien ne vous empêche d'écrire :

```
Au cas ou
   : (Expression_booléenne)
   : (Expression_booléenne)

   .
   .
   .

   : (Expression_booléenne)
      instruction(s)
Sinon
   instruction(s)
Fin de cas
```

ou :

```
Au cas ou
   : (Expression_booléenne)
   : (Expression_booléenne)
      instruction(s)
   .
   .
   .
   : (Expression_booléenne)
      instruction(s)
Sinon
Fin de cas
```

ou :

```
Au cas ou
Sinon
   instruction(s)
Fin de cas
```

Référence

Boucle...Fin de boucle, Conditions et boucles, Repeter...Jusque, Si...Sinon...Fin de si, Tant que...Fin tant que.

La syntaxe de la structure répétitive (ou boucle) Tant que...Fin tant que est la suivante :

```
Tant que (Expression_booléenne)
    instruction(s)
Fin tant que
```

Une boucle Tant que exécute les instructions comprises entre Tant que et Fin tant que aussi longtemps que l'expression booléenne est VRAIE. Elle teste l'expression booléenne initiale et n'entre pas dans la boucle (et donc n'exécute aucune instruction) si l'expression est à FAUX.

Il est utile d'initialiser la valeur testée dans l'expression booléenne juste avant d'entrer dans la boucle Tant que. Initialiser la valeur signifie lui affecter un contenu approprié, généralement pour que l'expression booléenne soit VRAIE et que le programme entre dans la boucle.

La valeur de l'expression booléenne doit pouvoir être modifiée par un élément situé à l'intérieur de la boucle, sinon elle s'exécutera indéfiniment. La boucle suivante est sans fin car Infini est toujours VRAI :

```
Infini:=Vrai
Tant que (Infini)
Fin tant que
```

Si vous vous retrouvez dans une telle situation (où une méthode s'exécute de manière incontrôlée), vous pouvez utiliser les fonctions de débogage de 4D et remonter à la source du problème. Pour plus d'informations sur ce point, reportez-vous à la section Débogage.

Exemple

```
    ` Est-ce que l'utilisateur veut ajouter un enregistrement?
CONFIRMER ("Ajouter un enregistrement?")
Tant que (OK = 1)   ` Tant que l'utilisateur accepte
    AJOUTER ENREGISTREMENT([Table])   ` Ajouter un nouvel enregistrement
Fin tant que   ` Une boucle Tant que se termine toujours par Fin tant que
```

Dans cet exemple, la valeur de la variable système OK est définie par la commande CONFIRMER avant que le programme n'entre dans la boucle. Si l'utilisateur clique sur le bouton OK dans la boîte de dialogue de confirmation, la variable OK prend la valeur 1 et la boucle est exécutée. Dans le cas contraire, la variable OK prend la valeur 0 et la boucle est ignorée. Une fois que le programme entre dans la boucle, la commande AJOUTER ENREGISTREMENT permet de continuer à l'exécuter car elle met la variable système OK à 1 lorsque l'utilisateur sauvegarde l'enregistrement.
Lorsque l'utilisateur annule (ne valide pas) le dernier enregistrement, la variable système OK prend la valeur 0 et la boucle s'arrête.

Référence

Au cas ou...Sinon...Fin de cas, Boucle...Fin de boucle, Conditions et boucles, Repeter...Jusque, Si...Sinon...Fin de si.

La syntaxe de la structure répétitive (ou boucle) Repeter...Jusque est la suivante :

```
Repeter
    instruction(s)
Jusque (Expression_booléenne)
```

La boucle Repeter...Jusque est semblable à la boucle Tant que...Fin tant que, à la différence qu'elle teste la valeur de l'expression booléenne après l'exécution de la boucle et non avant. Ainsi, la boucle est toujours exécutée au moins une fois, tandis que si l'expression booléenne est initialement à FAUX, la boucle Tant que...Fin tant que ne s'exécute pas du tout.

L'autre particularité de la boucle Repeter...Jusque est qu'elle se poursuit jusqu'à ce que l'expression booléenne soit à VRAI.

Exemple

Comparez l'exemple suivant avec celui de la boucle Tant que...Fin tant que : vous constatez qu'il n'est pas nécessaire d'initialiser l'expression booléenne — il n'y a pas de commande CONFIRMER pour initialiser la variable OK.

```
Repeter
    AJOUTER ENREGISTREMENT([aTable])
Jusque (OK=0)
```

Référence

Au cas ou...Sinon...Fin de cas, Boucle...Fin de boucle, Conditions et boucles, Si...Sinon...Fin de si, Tant que...Fin tant que.

La syntaxe de la structure répétitive Boucle...Fin de boucle est la suivante :

Boucle (Variable_Compteur; Expression_Début; Expression_Fin {; Expression_Incrément})
instructions(s)
Fin de boucle

La structure Boucle...Fin de boucle est une boucle contrôlée par un compteur :
• La variable compteur Variable_Compteur est une variable numérique (Réel, Entier ou Entier long) initialisée par Boucle...Fin de boucle à la valeur spécifiée par Expression_Début.
• La variable Variable_Compteur est incrémentée de la valeur spécifiée par le paramètre optionnel Expression_Incrément à chaque fois que la boucle est exécutée. Si vous ne passez pas de valeur dans Expression_Incrément, la variable compteur est incrémentée par défaut de un (1).
• Lorsque le compteur atteint la valeur définie par Expression_Fin, la boucle s'arrête.

Important : Les expressions numériques Expression_Début, Expression_Fin et Expression_Incrément sont évaluées une seule fois, au début de la boucle. Si ces expressions sont des variables, leur modification depuis l'intérieur de la boucle **n'affectera pas l'exécution de la boucle**.

Astuce : En revanche, vous pouvez, si vous le souhaitez, modifier la valeur de la variable Variable_Compteur depuis l'intérieur de la boucle et cela **affectera l'exécution de la boucle**.

• Généralement, Expression_Début est inférieure à Expression_Fin.
• Si les deux expressions sont égales, la boucle ne sera exécutée qu'une fois.
• Si Expression_Début est supérieure à Expression_Fin, la boucle ne s'exécutera pas du tout, à moins que vous ne spécifiiez une Expression_Incrément négative. Reportez-vous ci-dessous au paragraphe décrivant ce point.

Exemples élémentaires

(1) La boucle suivante s'exécute 100 fois :

Boucle (vCompteur;1;100)
` Faire quelque chose
Fin de boucle

(2) L'exemple suivant permet de traiter tous les éléments du tableau unTableau :

```
Boucle ($vlElem;1;Taille tableau(unTableau))
        ` Faire quelque chose avec l'élément
    unTableau{$vlElem}:=...
Fin de boucle
```

(3) L'exemple suivant permet d'examiner chaque caractère du texte vtDuTexte :

```
Boucle ($vlCar;1;Longueur(vtDuTexte))
        ` Faire quelque chose avec le caractère si c'est une tabulation
    Si (Code de caractere(vtDuTexte[[$vlCar]])=Caractere(Tab))
        ` ...
    Fin de si
Fin de boucle
```

(4) L'exemple suivant permet de traiter tous les enregistrements de la sélection de la table [uneTable]:

```
DEBUT SELECTION([uneTable])
Boucle ($vlEnrg;1;Enregistrements trouves([uneTable]))
        ` Faire quelque chose avec chaque enregistrement
    ENVOYER ENREGISTREMENT([uneTable])
        ` ...
        ` Passer à l'enregistrement suivant
    ENREGISTREMENT SUIVANT([uneTable])
Fin de boucle
```

La plupart des structures Boucle...Fin de boucle que vous écrirez dans vos bases ressembleront à celles présentées ci-dessus.

Décrémenter la variable Compteur

Dans certains cas, vous pouvez souhaiter disposer d'une boucle dont la valeur de la variable compteur décroît au lieu de croître. Pour cela, Expression_Début doit être supérieure à Expression_Fin et Expression_Increment doit être négative. Les exemples suivants effectuent les mêmes tâches que les précédents, mais en sens inverse :

(5) La boucle suivante s'exécute 100 fois :

```
Boucle (vCompteur;100;1;-1)
    ` Faire quelque chose
Fin de boucle
```

(6) L'exemple suivant permet de traiter tous les éléments du tableau unTableau :

```
Boucle ($vlElem;Taille tableau(unTableau);1;-1)
        ` Faire quelque chose avec l'élément
    unTableau{$vlElem}:=...
Fin de boucle
```

(7) L'exemple suivant permet d'examiner chaque caractère du texte vtDuTexte :

```
Boucle ($vlCar;Longueur(vtDuTexte);1;-1)
        ` Faire quelque chose avec le caractère si c'est une tabulation
    Si (Code de caractere(vtDuTexte[[$vlCar]])=Caractere(Tab))
        ` ...
    Fin de si
Fin de boucle
```

(8) L'exemple suivant permet de traiter tous les enregistrements de la sélection de la table [uneTable]:

```
ALLER A DERNIER ENREGISTREMENT([uneTable])
Boucle ($vlEnrg;Enregistrements trouves([uneTable]);1;-1)
        ` Faire quelque chose avec chaque enregistrement
    ENVOYER ENREGISTREMENT([uneTable])
        ` ...
        ` Passer à l'enregistrement précédent
    ENREGISTREMENT PRECEDENT([uneTable])
Fin de boucle
```

Incrementer la variable compteur de plus de 1

Si vous le souhaitez, vous pouvez passer dans Expression_Incrément une valeur (positive ou négative) dont la valeur absolue est supérieure à un.

(9) La boucle suivante ne traite que les éléments pairs du tableau unTableau :

```
Boucle ($vlElem;2;Taille tableau(unTableau);2)
        ` Faire quelque chose avec l'élément 2,4...2n
    unTableau{$vlElem}:=...
Fin de boucle
```

Sortir d'une boucle en modifiant la variable compteur
Dans certains cas, vous voudrez exécuter une boucle un certain nombre de fois, mais également pouvoir sortir si une autre condition devient Vraie.

Pour cela, il vous suffit de tester cette condition à l'intérieur de la boucle et, si elle devient Vraie, de "forcer" la valeur de la variable compteur, de manière à ce qu'elle soit supérieure à celle de Expression_Fin.

(10) Dans l'exemple suivant, vous effectuez une boucle parmi les enregistrements d'une sélection jusqu'à ce que la fin de la sélection soit atteinte, ou bien jusqu'à ce que la variable interprocess <>vbStop, initialement fixée à Faux, prenne la valeur Vrai. Cette variable est gérée par une méthode projet APPELER SUR EVENEMENT utilisée pour interrompre l'opération :

```
<>vbStop:=Faux
APPELER SUR EVENEMENT ("GESTION STOP")
    ` GESTION STOP définit <>vbStop à Vrai si les touches Ctrl+point (Windows) ou
                                                    Commande+point (Mac OS) sont enfoncées
$vlNbEnrgs:=Enregistrements trouves([aTable])
DEBUT SELECTION([aTable])
Boucle ($vlEnrgs;1;$vlNbEnrgs)
        ` Faire quelque chose avec l'enregistrement
    ENVOYER ENREGISTREMENT([aTable])
        ` ...
        ` Aller à l'enregistrement suivant
    Si (<>vbStop)
        $vlEnrgs:=$vlNbEnrgs+1 ` Forcer la variable compteur à stopper la boucle
    Sinon
        ENREGISTREMENT SUIVANT([aTable])
    Fin de si
Fin de boucle
APPELER SUR EVENEMENT("")
Si (<>vbStop)
    ALERTE("L'opération a été interrompue.")
Sinon
    ALERTE("L'opération s'est terminée avec succès.")
Fin de si
```

Comparaison des structures répétitives

Reprenons le premier exemple employé pour la structure Boucle...Fin de boucle :

(1) La boucle suivante s'exécute 100 fois :

```
Boucle (vCompteur;1;100)
    ` Faire quelque chose
Fin de boucle
```

Il est intéressant d'examiner la manière dont les boucles Tant que...Fin tant que et Repeter...Jusque effectuent la même action :

Voici la boucle Tant que...Fin tant que équivalente :

```
$i := 1 ` Initialisation du compteur
Tant que ($i<=100) ` Boucle 100 fois
    ` Faire quelque chose
    $i := $i + 1 ` Il faut incrémenter le compteur
Fin tant que
```

Voici la boucle Repeter...Jusque équivalente :

```
$i := 1 ` Initialisation du compteur
Repeter
    ` Faire quelque chose
    $i := $i + 1 ` Il faut incrémenter le compteur
Jusque ($i=100) ` Boucle 100 fois
```

Astuce : La boucle Boucle...Fin de boucle est généralement plus rapide que les boucles Tant que...Fin tant que et Repeter...Jusque car 4D teste la condition en interne pour chaque cycle de la boucle et incrémente lui-même le compteur. Par conséquent, nous vous conseillons de préférer à chaque fois que c'est possible la structure Boucle...Fin de boucle.

Optimiser l'exécution de Boucle...Fin de boucle

Vous pouvez utiliser comme compteur une variable interprocess, process ou locale, et lui attribuer le type Réel, Entier ou Entier long. Pour des boucles longues, et particulièrement en mode compilé, nous vous conseillons d'employer des variables locales de type Entier long.

(11) Voici un exemple :

```
C_ENTIER LONG($vlCompteur)   ` Utilisons une variable locale de type Entier long
Boucle ($vlCompteur;1;10000)
    ` Faire quelque chose
Fin de boucle
```

Structures Boucle...Fin de boucle **emboîtées**

Vous pouvez emboîter autant de structures répétitives que vous voulez (dans les limites du raisonnable). Cela s'applique aux structures de type Boucle...Fin de boucle. Il y a dans ce cas une erreur courante à éviter : assurez-vous d'utiliser une variable compteur différente par structure de boucle. Voici deux exemples :

(12) L'exemple suivant permet de traiter tous les éléments d'un tableau à deux dimensions :

```
Boucle ($vlElem;1;Taille tableau(unTableau))
        ` ...
        ` Faire quelque chose avec la ligne
        ` ...
    Boucle ($vlSousElem;1;Taille tableau(unTableau{$vlElem}))
        ` Faire quelque chose avec l'élément
        unTableau{$vlElem}{$vlSousElem}:=...
    Fin de boucle
Fin de boucle
```

(13) L'exemple suivant construit un tableau de pointeurs vers tous les champs de type Date présents dans la base :

```
TABLEAU POINTEUR($apChampsDate;0)
$vlElem:=0
Boucle ($vlTable;1;Lire numero derniere table)
    Si(Est un numero de table valide($vlTable))
        Boucle($vlChamp;1;Lire numero dernier champ($vlTable))
            Si(Est un numero de champ valide($vlTable;$vlChamp))
                $vpChamp:=Champ($vlTable;$vlChamp)
                Si (Type($vpChamp>)=Est une date)
                    $vlElem:=$vlElem+1
                    INSERER DANS TABLEAU($apChampsDate;$vlElem)
                    $apChampsDate{$vlElem}:=$vpChamp
                Fin de si
                        Fin de si
```

```
            Fin de boucle
        Fin de si
    Fin de boucle
```

Référence

Au cas ou...Sinon...Fin de cas, Conditions et boucles, Repeter...Jusque, Si...Sinon...Fin de si, Tant que...Fin tant que.

Pour faire fonctionner les commandes, les opérateurs, et les autres composants du langage, vous les placez dans des méthodes. Ce chapitre décrit les fonctionnalités communes à tous les types de méthodes. Il existe plusieurs types de méthodes : les méthodes objet, les méthodes formulaire, les méthodes table (ou triggers), les méthodes projet et les méthodes base.

Une méthode est composée de plusieurs **lignes d'instructions**. Une ligne d'instructions effectue une action. Cette ligne d'instruction peut être simple ou complexe. Cette ligne peut être aussi longue que vous voulez (elle peut comporter jusqu'à 32 000 caractères, ce qui est normalement suffisant pour la plupart des instructions).

Par exemple, la ligne suivante est une instruction qui ajoute un nouvel enregistrement à la table [Personnes] :

```
AJOUTER ENREGISTREMENT([Personnes])
```

Une méthode contient également des **tests** et des **boucles** qui structurent son exécution. Pour plus d'informations sur les structures de programmation, reportez-vous à la section Conditions et boucles.

Note : La taille maximale d'une méthode est limitée à 2 Go de texte ou 32 000 lignes d'instructions. Au-delà de ces limites, un message d'alerte apparaît, indiquant que les lignes supplémentaires ne seront pas affichées.

Types de méthodes

Il existe cinq types de méthodes dans 4D :

• **Méthodes objet** : une méthode objet est une courte méthode associée à un objet actif dans un formulaire. En général, les méthodes objet "gèrent" l'objet au moment de l'affichage ou de l'impression du formulaire. Vous ne pouvez pas appeler une méthode objet, 4D l'exécute automatiquement lorsqu'un événement implique l'objet auquel la méthode est rattachée.

• **Méthodes formulaire** : Une méthode formulaire est associée à un formulaire. Vous pouvez utiliser une méthode formulaire pour gérer les données et les objets, mais il est généralement plus simple et plus efficace d'utiliser des méthodes objet dans ce cas. Vous ne pouvez pas appeler une méthode formulaire, 4D gère automatiquement son exécution lorsqu'un événement implique le formulaire auquel la méthode est attachée.

• **Méthodes table/triggers** : Un trigger est associé à une table. Vous ne pouvez pas appeler un trigger. 4D les exécute automatiquement à chaque opération élémentaire effectuée sur les enregistrements de la table (Chargement, Ajout, Suppression et Modification). Les triggers sont des méthodes pouvant prévenir toute action "illégale" opérée sur les enregistrements. Par exemple, dans un système de facturation, vous pouvez empêcher les utilisateurs d'ajouter des factures sans avoir spécifié le client concerné. Les triggers constituent un outil très puissant pour contrôler les opérations effectuées sur les tables ainsi que pour prévenir toute perte de données accidentelle. Vous pouvez écrire des triggers très simples ou très sophistiqués.

Pour plus d'informations sur les triggers, reportez-vous à la section Présentation des triggers.

• **Méthodes projet** : A la différence des précédents types de méthodes, les méthodes projet ne sont associées à aucun objet, formulaire ou table, elles peuvent être utilisées à n'importe quel endroit de votre base. Les méthodes projet sont réutilisables et disponibles à tout moment, pour toute autre méthode. Si vous devez répéter certaines tâches, vous n'avez pas besoin de réécrire plusieurs méthodes identiques dans chaque cas. Vous pouvez appeler une méthode projet partout où vous en avez besoin — depuis d'autres méthodes projet, objet ou formulaire. Lorsque vous appelez une méthode projet, elle se comporte comme si vous l'aviez écrite à l'endroit d'où vous l'appelez. Les méthodes projet utilisées par d'autres méthodes sont appelées des **sous-routines**. Une méthode projet qui retourne un résultat peut aussi être appelée une **fonction**.

Il existe une autre manière d'appeler les méthodes projet : il suffit de les associer à des commandes de menus. Lorsque vous associez une méthode projet à une commande de menu, la méthode est exécutée lorsque la commande du menu est sélectionnée par un utilisateur.

• **Méthodes base** : Tout comme les méthodes formulaire ou objet sont exécutées lorsqu'un événement se produit dans un formulaire, il existe des méthodes associées à la base entière, et qui sont exécutées lorsqu'un événement de session de travail se produit. Ce sont les **méthodes base**. Par exemple, à chaque fois que vous ouvrez la base, vous pouvez vouloir initialiser des variables qui seront utilisées pendant toute la session de travail. Pour cela, vous pouvez écrire des instructions dans la Méthode base Sur ouverture, exécutée automatiquement par 4D lorsque vous lancez la base.

Pour plus d'informations sur les méthodes base, reportez-vous à la section Présentation des méthodes base.

Un exemple de méthode projet

Toutes les méthodes sont fondamentalement identiques — elles débutent sur la première ligne et traitent chaque ligne d'instruction jusqu'à ce qu'elles atteignent la dernière ligne (c'est-à-dire qu'elles s'exécutent séquentiellement).

Voici un exemple de méthode projet :

```
CHERCHER([Personnes])      ` Afficher l'éditeur de recherches
Si (OK=1)   ` L'utilisateur a cliqué sur OK (et non sur Annuler)
    Si (Enregistrements trouves ([Personnes])=0) ` Si aucun enregistrement n'est trouvé…
        AJOUTER ENREGISTREMENT([Personnes])
            ` Permettre à l'utilisateur d'ajouter un enregistrement
    Fin de si
Fin de si     ` Fin
```

Chaque ligne de l'exemple est une **ligne d'instruction**. Tout ce que vous écrivez dans le langage de 4D est appelé du code. Le code est exécuté — ce qui signifie que 4D effectue l'action spécifiée par le code.

Nous allons examiner très attentivement la première ligne ; nous irons plus rapidement par la suite :

```
CHERCHER([Personnes])      ` Afficher l'éditeur de recherches
```

Le premier mot de la ligne, CHERCHER, est une commande. Une commande est un élément du langage de 4D — elle effectue une action. Dans ce cas, CHERCHER affiche l'éditeur de recherches, tout comme le fait la commande **Chercher...** dans le menu **Enregistrements** en mode Développement.

Les parenthèses signifient qu'un **paramètre** est **passé** à la commande CHERCHER. Un paramètre (ou **argument**) est une valeur nécessaire à une commande pour remplir son rôle. Dans ce cas, [Personnes] est le nom d'une table. Les noms des tables sont toujours écrits entre crochets ([…]). Nous pouvons donc dire : « La table Personnes est un paramètre de la commande CHERCHER ». Une commande qui accepte plusieurs paramètres.

Enfin, il y a un **commentaire** à la fin de la ligne. Un commentaire vous informe (ainsi que toute personne qui examinera votre méthode) de ce qui se passe dans le code. Un commentaire est signalé par une apostrophe inversée (`). Dans une ligne, tout ce qui suit un signe de commentaire est ignoré lorsque le code est exécuté. Un commentaire peut être placé sur sa propre ligne, ou à la suite du code, comme dans l'exemple. N'hésitez pas à utiliser les commentaires dans vos méthodes ; la lecture et la compréhension de votre code sont facilitées, aussi bien pour vous-même que pour d'autres personnes.

Note : Un commentaire peut contenir jusqu'à 32 000 caractères.

La ligne suivante de notre exemple vérifie qu'aucun enregistrement n'a été trouvé :

```
    Si (Enregistrements trouves ([Personnes])=0) ` Si aucun enregistrement n'est trouvé…
```

La ligne d'instruction Si est un **test conditionnel** — une instruction qui contrôle l'exécution au pas à pas de votre méthode. Le Si effectue un test, et si le test est VRAI, la ou les lignes suivantes sont exécutées. Enregistrements trouves est une fonction — c'est-à-dire une commande qui retourne une valeur. Ici, Enregistrements trouves retourne le nombre d'enregistrements de la sélection courante de la table passée en paramètre.

Note : Seule la première lettre du nom d'une fonction est en majuscule. C'est la convention d'écriture utilisée pour les fonctions de 4D.

Vous savez déjà ce qu'est la sélection courante : le groupe d'enregistrements avec lequel vous travaillez à un instant donné. Si le nombre d'enregistrements est égal à 0 (c'est-à-dire, si aucun enregistrement n'a été trouvé), la ligne de code suivante est exécutée :

```
AJOUTER ENREGISTREMENT([Personnes])
```

La commande AJOUTER ENREGISTREMENT affiche un formulaire pour permettre à l'utilisateur de créer un nouvel enregistrement. Notez que cette ligne comporte une indentation. 4D formate votre code automatiquement ; l'indentation vous permet d'identifier facilement ce qui est dépendant du test conditionnel (Si).

```
Fin de si      ` Fin
```

La ligne d'instruction Fin de si conclut la séquence d'instructions contrôlée par le test Si. Pour chaque ligne d'instruction de test conditionnel, votre code doit comporter une instruction correspondante indiquant au langage où s'arrête le test. Nous vous conseillons de bien maîtriser les concepts évoqués dans ce chapitre. S'ils sont nouveaux pour vous, n'hésitez pas à relire les explications fournies jusqu'à ce qu'elles vous semblent claires.

Pour en savoir plus...

• Pour plus d'informations sur les méthodes objet et formulaire, reportez-vous à la description de la commande Evenement formulaire ainsi qu'au manuel *Mode Développemen*t de 4D.
• Pour travailler avec les triggers, reportez-vous à la section Présentation des triggers.
• Pour une description détaillée des méthodes projet, reportez-vous à la section Méthodes projet.
• Les méthodes base sont détaillées dans la section Présentation des méthodes base et ses sous-sections.

Référence

Conditions et boucles, Constantes, Nommer les objets du langage 4D, Opérateurs, Pointeurs, Présentation des méthodes base, Présentation des tableaux, Présentation des triggers, Types de données, Variables.

Les méthodes projet, comme leur nom l'indique, s'appliquent à la totalité de votre projet, c'est-à-dire votre base de données. Alors que les méthodes formulaire ou les méthodes objet par exemple sont associées à des formulaires ou des objets, une méthode projet est disponible partout — elle n'est associée à aucun élément particulier de la base. Une méthode projet peut tenir les rôles suivants, en fonction de la manière dont elle est exécutée et utilisée :

- Méthode de menu
- Sous-routine et fonction
- Méthode de gestion de process
- Méthode de gestion d'événements
- Méthode de gestion d'erreurs

Ces appellations ne qualifient pas la nature des méthodes (ce qu'elles sont), mais leur fonction (ce qu'elles font).

Une **méthode de menu** est une méthode projet appelée depuis une commande de menu personnalisé. Elle se comporte comme un agent de police chargé de la circulation, dirigeant les flux de votre application. Les méthodes de menu contrôlent, aiguillent lorsque c'est nécessaire, affichent les formulaires, génèrent des états ou encore gèrent votre base.

Le deuxième type de méthode projet peut être considéré comme une méthode asservie — d'autres méthodes lui demandent d'effectuer des tâches. Ce type de méthode est appelé **sous-routine**. Une sous-routine qui retourne une valeur est appelée une **fonction**.

Une **méthode de gestion de process** est une méthode projet appelée lorsqu'un process est démarré. Le process existera tant que la méthode sera en cours d'exécution. Pour plus d'informations sur les process, reportez-vous à la section Introduction aux process. A noter qu'une méthode de menu associée à une commande de menu pour laquelle la propriété **Démarrer un nouveau process** est sélectionnée, est aussi la méthode de gestion de process pour le process créé.

Une **méthode de gestion d'événements** est une méthode dédiée à la gestion des événements, qui s'exécute dans un process différent de celui de la méthode de gestion des process. Généralement, pour la gestion des événements, vous pouvez laisser 4D faire le gros du travail. Par exemple, lors de la saisie de données, 4D détecte les clics souris et les touches enfoncées, puis appelle les méthodes objet et formulaire correspondantes, vous permettant ainsi de prévoir dans ces méthodes les traitements appropriés aux événements.

Dans d'autres circonstances, vous devez gérer directement les événements. Si, par exemple, vous exécutez une opération de longue durée (telle qu'une Boucle...Fin de boucle appliquée à chaque enregistrement), vous souhaitez pouvoir interrompre l'opération en tapant la touche **Esc**. Dans ce cas, une méthode de gestion d'événements est parfaitement adaptée. Pour plus d'informations, reportez-vous à la description de la commande APPELER SUR EVENEMENT.

Une **méthode de gestion d'erreurs** est une méthode projet d'interruption. Elle s'exécute à l'intérieur du process dans lequel elle a été installée à chaque fois qu'une erreur se produit. Pour plus d'informations, reportez-vous à la description de la commande APPELER SUR ERREUR.

Méthodes de menu

Une méthode de menu est appelée en mode Application lorsque la commande de menu personnalisé à laquelle elle est associée est sélectionnée. Vous assignez la méthode à la commande de menu dans l'éditeur de menus de 4D. Lorsque l'utilisateur sélectionne la commande de menu, la méthode est exécutée. Ce fonctionnement est l'un des principaux aspects de la personnalisation d'une base de données. C'est en créant des menus qui appellent des méthodes de menu que vous personnalisez votre base. Reportez-vous au manuel *Mode Développement* de 4D pour plus d'informations sur l'éditeur de menus.

Les commandes de menus personnalisés peuvent déclencher une ou plusieurs actions. Par exemple, une commande de menu de saisie d'enregistrements peut appeler une méthode effectuant deux actions : afficher le formulaire entrée approprié et appeler la commande AJOUTER ENREGISTREMENT jusqu'à ce que l'utilisateur annule la saisie de nouveaux enregistrements.

L'automatisation de séquences d'actions est une possibilité très puissante du langage de programmation de 4D. A l'aide des menus personnalisés, vous pouvez automatiser des séquences de tâches, vous permettez aux utilisateurs de naviguer plus facilement dans votre base.

Sous-routines

Lorsque vous avez écrit une méthode projet, elle devient partie intégrante du langage de la base dans laquelle elle a été créée. Vous pouvez alors l'appeler de la même manière que vous appelez les commandes intégrées de 4D. Une méthode projet utilisée de cette manière est appelée une sous-routine.

L'utilisation de sous-routines procure les avantages suivants :

- Réduction du code répétitif,
- Clarification des méthodes,
- Modification plus facile des méthodes,
- Création de code modulaire.

Imaginons par exemple que vous travaillez avec une base de clients. A mesure que vous construisez la base, vous vous apercevez que vous répétez souvent certaines tâches, telles que la recherche d'un client et la modification de son enregistrement. Le code nécessaire à l'accomplissement de cette opération pourrait être :

```
    ` Recherche d'un client
CHERCHER PAR EXEMPLE([Clients])
    ` Sélection du formulaire entrée
FORMULAIRE ENTREE([Clients];"Saisie de données")
    ` Modification de l'enregistrement du client
MODIFIER ENREGISTREMENT([Clients])
```

Si vous n'utilisez pas de sous-routines, vous devrez écrire ce code à chaque fois que vous voudrez modifier l'enregistrement d'un client. Si cette opération peut être réalisée dans dix endroits différents de votre base, vous devrez la réécrire dix fois. Grâce aux sous-routines, vous ne l'écrirez qu'une seule fois en tout. C'est le premier avantage des sous-routines : réduire la quantité de code à écrire.

Si le code ci-dessus était une méthode projet appelée MODIFIER CLIENT, vous l'exécuteriez simplement en inscrivant son nom dans une autre méthode. Par exemple, pour modifier l'enregistrement d'un client puis l'imprimer, vous n'auriez qu'à écrire :

```
MODIFIER CLIENT
IMPRIMER SELECTION([Clients])
```

Cette possibilité simplifie énormément vos méthodes. Dans l'exemple ci-dessus, il n'est pas nécessaire de savoir comment fonctionne la méthode MODIFIER CLIENT, mais uniquement ce qu'elle fait. C'est le deuxième avantage que vous pouvez tirer de l'utilisation de sous-routines : la clarification de votre code. Ainsi, ces méthodes deviennent en quelque sorte des extensions du langage de 4D.

Si vous devez modifier votre mode de recherche des clients, comme dans notre exemple, il vous suffit de modifier une seule méthode, et non dix. C'est un autre avantage des sous-routines : faciliter les modifications de votre code.

Avec les sous-routines, vous rendez votre code modulaire. Cela signifie simplement que vous dissociez votre code en modules (sous-routines), chacun d'entre eux effectuant une tâche logique. Examinez le code suivant, tiré d'une base de gestion de comptes chèques :

```
CHERCHER CHEQUES EMIS   ` Rechercher les chèques émis
RAPPROCHER COMPTE   ` Rapprocher le compte
IMPRIMER RELEVE   ` Imprimer un relevé
```

Même pour quelqu'un qui ne connaît pas la base, le code est clair. Il n'est pas nécessaire d'examiner chaque sous-routine. Elles peuvent contenir de nombreuses lignes d'instructions et effectuer des opérations complexes, mais l'important est ce qu'elles font.

Nous vous conseillons de découper votre code en tâches logiques, ou modules, à chaque fois que c'est possible.

Passer des paramètres aux méthodes

Vous aurez souvent besoin de fournir des valeurs à vos méthodes. Vous pouvez facilement effectuer cette opération grâce aux paramètres.

Les **paramètres** (ou arguments) sont des données dont les méthodes ont besoin pour s'exécuter — le terme "paramètres" ou "arguments" est utilisé indifféremment dans ce manuel. Des paramètres sont également passés aux commandes intégrées de 4D. Dans l'exemple ci-dessous, la chaîne "Bonjour" est un paramètre de la commande ALERTE :

```
ALERTE("Bonjour")
```

Les paramètres sont passés de la même manière aux méthodes. Par exemple, si la méthode FAIRE QUELQUE CHOSE accepte trois paramètres, l'appel à cette méthode pourrait être de la forme suivante :

```
FAIRE QUELQUE CHOSE(AvecCeci;EtCela;CommeCeci)
```

Les paramètres sont séparés par des points-virgules (;).

Dans la sous-routine (la méthode appelée), la valeur de chaque paramètre est automatiquement copiée séquentiellement dans des variables locales numérotées : $1, $2, $3, etc. La numérotation des variables locales représente l'ordre des paramètres.

Ces variables/paramètres locaux ne sont pas réellement les champs, variables ou expressions passées à la **méthode appelante** : elles contiennent simplement leurs valeurs.

A l'intérieur de la sous-routine, vous pouvez utiliser les paramètres $1, $2... de la même manière que vous utilisez les autres variables locales.

Note : Toutefois, dans le cas où vous utilisez des commandes qui modifient la valeur de la variable passée en paramètre (par exemple Trouver dans champ), les paramètres $1, $2... ne peuvent pas être utilisés directement. Vous devez d'abord les recopier dans des variables locales standard (par exemple $mavar:=$1).

Puisque ces variables sont locales, elles ne sont définies qu'à l'intérieur de la sous-routine et sont effacées à la fin de son exécution. Pour cette raison, une sous-routine ne peut pas modifier, au niveau de la méthode appelante, la valeur réelle des champs ou des variables passé(e)s en paramètre. Par exemple :

```
    ` Voici une partie de la méthode MA METHODE
    ` ...
FAIRE QUELQUE CHOSE ([Personnes]Nom)  ` Admettons que [Personnes]Nom est égal à "william"
ALERTE([Personnes]Nom)

    ` Voici le code de la méthode FAIRE QUELQUE CHOSE
$1:=Majusc($1)
ALERTE($1)
```

La boîte de dialogue d'alerte affichée par FAIRE QUELQUE CHOSE contiendra "WILLIAM" et celle affichée par MA METHODE contiendra "william". La méthode a modifié localement la valeur du paramètre $1, mais cela n'affecte pas la valeur du champ [Personnes]Nom passé en paramètre par la méthode MA METHODE.

Si vous voulez réellement que la méthode FAIRE QUELQUE CHOSE modifie la valeur du champ, deux solutions s'offrent à vous :

1. Plutôt que de passer le champ à la méthode, vous lui passez un pointeur :

```
    ` Voici une partie de la méthode MA METHODE
    ` ...
FAIRE QUELQUE CHOSE (->[Personnes]Nom) ` Admettons que [Personnes]Nom est égal à "william"
ALERTE([Personnes]Nom)

    ` Voici le code de la méthode FAIRE QUELQUE CHOSE
$1->:=Majusc($1->)
ALERTE($1->)
```

Ici, le paramètre n'est pas le champ lui-même, mais un pointeur vers le champ. Ainsi, à l'intérieur de la méthode FAIRE QUELQUE CHOSE, $1 ne contient plus la valeur du champ mais un pointeur vers le champ. L'objet **référencé** par $1 ($1-> dans le code ci-dessus) est le champ lui-même. Par conséquent, la modification de l'objet référencé dépasse les limites de la sous-routine et le champ lui-même est affecté. Dans cet exemple, les deux boîtes de dialogue d'alerte afficheront "WILLIAM".

Pour plus d'informations sur les pointeurs, reportez-vous à la section Pointeurs.

2. Plutôt que la méthode FAIRE QUELQUE CHOSE “fasse quelque chose”, vous pouvez la réécrire de manière à ce qu'elle retourne une valeur :

```
    ` Voici une partie de la méthode MA METHODE
    ` ...
[Personnes]Nom:=FAIRE QUELQUE CHOSE ([Personnes]Nom) ` Admettons que [Personnes]Nom est égal à "william"
ALERTE([Personnes]Nom)

    ` Voici le code de la méthode FAIRE QUELQUE CHOSE
$0:=$1
ALERTE($0)
```

Une sous-routine retournant une valeur est appelée une fonction. Ce point est traité dans les paragraphes suivants.

Note de programmation avancée : Les paramètres à l'intérieur de la sous-routine sont accessibles par l'intermédiaire des variables locales $1, $2... De plus, des paramètres peuvent être optionnels et être référencés à l'aide de la syntaxe ${...}. Pour plus d'informations sur ce point, reportez-vous à la description de la commande Nombre de parametres.

Les fonctions, méthodes projet retournant une valeur

Les méthodes peuvent retourner des valeurs. Une méthode qui retourne une valeur est appelée une **fonction**.

Les commandes de 4D ou de plug-ins qui retournent une valeur sont également appelées fonctions.

Par exemple, la ligne d'instruction suivante utilise une fonction intégrée, Longueur, qui retourne la longueur d'une chaîne. La valeur retournée par Longueur est placée dans une variable appelée MaLongueur :

```
MaLongueur:=Longueur("Comment suis-je arrivé là ?")
```

Toute sous-routine peut retourner une valeur. La valeur à retourner est placée dans la variable locale $0.

Par exemple, la fonction suivante, appelée Majuscules4, retourne une chaîne dont les quatre premiers caractères ont été passés en majuscules :

```
$0:=Majusc(Sous chaine($1; 1; 4))+Sous chaine($1; 5)
```

Voici un exemple qui utilise la fonction Majuscules4 :

```
NouvellePhrase:=Majuscules4 ("Bien joué.")
```

Dans ce cas, la variable NouvellePhrase prend la valeur "BIEN joué."

Le **retour de fonction**, $0, est une variable locale à la sous-routine. Elle peut être utilisée en tant que telle à l'intérieur de la sous-routine. Par exemple, dans le cas de la méthode FAIRE QUELQUE CHOSE utilisée précédemment, $0 recevait d'abord la valeur de $1, puis était utilisée en tant que paramètre de la commande ALERTE. Dans une sous-méthode, vous pouvez utiliser $0 comme n'importe quelle autre variable locale. C'est 4D qui retourne sa valeur finale (sa valeur courante au moment où la sous-routine se termine) à la méthode appelée.

Méthode projet récursives

Des méthodes projet peuvent s'appeler les unes les autres, et en particulier :
• Une méthode A peut appeler une méthode B, qui appelle A, donc A appelle B de nouveau, etc.
• Une méthode peut s'appeler elle-même.

Cela s'appelle la **récursivité**. Le langage de 4D supporte pleinement la récursivité.

Examinons l'exemple suivant : vous disposez d'une table [Amis et relations] composée de l'ensemble de champs suivant (très simplifié) :
- [Amis et parents]Nom
- [Amis et parents]Enfant'Nom

Pour cet exemple, nous supposons que les valeurs des champs sont uniques (il n'existe pas deux personnes avec le même nom). A partir d'un nom, vous voulez écrire la phrase “Un de mes amis, Pierre, qui est le rejeton de Paul qui est le rejeton de Martine qui est le rejeton de Robert qui est le rejeton de Gertrude, fait cela pour gagner sa vie !” :

1. Vous pouvez procéder de la manière suivante :

```
$vsNom:=Demander("Saisissez le nom :";"Pierre")
Si (OK=1)
   CHERCHER([Amis et parents];[Amis et parents]Nom=$vsNom)
   Si (Enregistrements trouves([Amis et parents])>0)
      $vtHistoireComplète:="Un de mes amis, "+$vsNom
      Repeter
         CHERCHER([Amis et parents];[Amis et parents]Enfant'Nom=$vsNom)
         $vlResultRecherche:=Enregistrements trouves([Amis et parents])
         Si ($vlResultRecherche>0)
            $vtHistoireComplète:=$vtHistoireComplète+" qui est le rejeton de "+
                                                                  [Amis et parents]Nom
            $vsNom:=[Amis et parents]Nom
         Fin de si
      Jusque ($vlResultRecherche=0)
      $vtHistoireComplète:=$vtHistoireComplète+", fait cela pour gagner sa vie !"
      ALERTE($vtHistoireComplète)
   Fin de si
Fin de si
```

2. Vous pouvez également procéder ainsi :

```
$vsNom:=Demander("Saisissez le nom :";"Pierre")
Si (OK=1)
   CHERCHER([Amis et parents];[Amis et parents]Nom=$vsNom)
   Si (Enregistrements trouves([Amis et parents])>0)
      ALERTE("Un de mes amis, "+Généalogie de ($vsNom)+", fait cela pour gagner sa
                                                                        vie !")
   Fin de si
Fin de si
```

en utilisant la fonction récursive Généalogie de suivante :

```
      ` Méthode projet Généalogie de
      ` Généalogie de ( Chaîne ) -> Texte
      ` Généalogie de ( Nom ) -> Partie de la phrase

$0:=$1
CHERCHER([Amis et parents];[Amis et parents]Enfant'Nom=$1)
Si (Enregistrements trouves([Amis et parents])>0)
   $0:=$0+" qui est le rejeton de "+Généalogie de ([Amis et parents]Nom)
Fin de si
```

Vous notez que la méthode Généalogie de s'appelle elle-même.

La première manière de procéder utilise un **algorithme itératif**. La seconde manière utilise un **algorithme récursif**.

Lorsque vous implémentez du code pour traiter des cas comme celui décrit ci-dessus, vous aurez toujours le choix entre écrire des méthodes utilisant des algorithmes itératifs ou récursifs. Généralement, la récursivité permet d'écrire du code plus concis, lisible et plus facilement modifiable, mais utiliser la récursivité n'est absolument pas obligatoire.

Dans 4D, la récursivité est typiquement utilisée pour :
• Traiter les enregistrements de tables liées les unes aux autres de la même manière que décrit dans l'exemple ci-dessus.
• Naviguer parmi les documents et les dossiers de votre disque à l'aide des commandes LISTE DES DOSSIERS et LISTE DES DOCUMENTS. Un dossier peut contenir des dossiers et des documents, les sous-dossiers peuvent eux-mêmes contenir des dossiers et des documents, etc.

Important : Les appels récursifs doivent toujours se terminer à un moment donné. Dans l'exemple ci-dessus, la méthode Généalogie de cesse de s'appeler elle-même lorsque la recherche ne trouve plus d'enregistrement. Sans ce test conditionnel, la méthode s'appellerait indéfiniment et 4D pourrait au bout d'un certain temps retourner l'erreur “La pile est pleine” car le programme n'aurait plus assez de place pour "empiler" les appels (ainsi que les paramètres et les variables locales utilisés dans la méthode).

Référence

Conditions et boucles, Méthodes, Présentation des méthodes base.

3

BLOB

Définition

4D prend en charge les données de type BLOB (Binary Large OBjects).

Vous pouvez définir des champs de type BLOB et des variables de type BLOB :

• Pour créer un champ de type BLOB, sélectionnez BLOB dans la liste déroulante **Type de champ** dans la fenêtre des **Propriétés de champ**.
• Pour créer une variable de type BLOB, utilisez la directive de compilation C_BLOB. Vous pouvez créer des variables BLOB locales, process et interprocess.

Note : Les tableaux ne peuvent avoir le type BLOB.

Dans 4D, un BLOB est une série contiguë d'octets de longueur variable qui peut être traitée comme un seul objet ou dont les octets peuvent être adressés individuellement. Un BLOB peut être vide (longueur nulle) ou contenir jusqu'à 2147483647 octets (2 Go).

Les BLOBs et la mémoire

Lorsque vous travaillez avec un BLOB, il est stocké entièrement en mémoire. Si vous travaillez avec une variable de type BLOB, elle n'existe qu'en mémoire. Si vous travaillez avec un champ de type BLOB, il est chargé en mémoire à partir du disque, comme le reste de l'enregistrement auquel il appartient.

A l'instar des autres types de champs pouvant contenir une grande quantité de données (comme les champs de type Image et Sous-table), les champs de type BLOB ne sont pas dupliqués en mémoire lorsque vous modifiez un enregistrement. Par conséquent, les résultats renvoyés par Ancien et Modifie ne sont pas significatifs lorsque ces fonctions sont appliquées à des champs de type BLOB.

Afficher des BLOBs

Comme un BLOB peut contenir n'importe quel type de données, il n'existe pas de mode de représentation à l'écran par défaut des BLOBs. Si vous affichez un champ ou une variable de type BLOB dans un formulaire, l'objet sera toujours vide, quel que soit son contenu.

Champs de type BLOB

Vous pouvez utiliser des champs de type BLOB pour stocker tout type de données dont la taille est inférieure ou égale à 2 Giga-octets. Vous ne pouvez pas indexer un champ de type BLOB. Si vous voulez rechercher des enregistrements à partir d'une valeur contenue dans un champ BLOB, il sera nécessaire d'écrire une formule.

Passage des paramètres, pointeurs et résultats de fonctions

Les BLOBs dans 4D peuvent être passés comme paramètres aux commandes 4D ou aux routines des plug-ins qui attendent un paramètre de type BLOB. Les BLOBs peuvent également être passés aux méthodes que vous créez ou être retournés comme un résultats de fonctions. Pour passer un BLOB à une de vos méthodes, vous pouvez aussi définir un pointeur vers le BLOB et passer le pointeur comme paramètre. Voici quelques exemples :

```
      ` Déclarer une variable de type BLOB
C_BLOB (touteVarBLOB)
      ` Le BLOB est passé comme paramètre à une commande 4D
FIXER TAILLE BLOB (touteVarBLOB;1024*1024)
      ` Le BLOB est passé comme paramètre à une routine externe
$CodeErr:= Faites_Quelque_chose_avec_ce_BLOB  (touteVarBLOB)
      ` Le BLOB est passé comme paramètre à une méthode qui retourne un BLOB
C_BLOB (recupBlob)
recupBlob:=Remplir_Blob (touteVarBLOB)
      ` Un pointeur vers le BLOB est passé comme paramètre à une de vos méthodes
REMPLIR BLOB AVEC DES ZEROS (->touteVarBLOB )
```

Note pour les développeurs de plug ins 4D : Un paramètre de type BLOB se déclare “&O” (la lettre “O” et non le chiffre “0”).

Affectation

Vous pouvez affecter la valeur d'un BLOB à d'autres BLOBs, comme dans l'exemple suivant :

```
      ` Déclarer deux variables de type BLOB
C_BLOB (vBlobA;vBlobB)
      ` Fixer la taille du premier BLOB à 10Ko
FIXER TAILLE BLOB (vBlobA;10*1024)
      ` Affectez le premier BLOB au second
vBlobB:=vBlobA
```

En revanche, il n'existe pas d'opérateur pouvant être utilisé avec des BLOBs ; il n'existe pas d'expression de type BLOB.

Adresser le contenu d'un BLOB

Chaque octet d'un BLOB peut être adressé individuellement, à l'aide des accolades {...}. Dans un BLOB, les octets sont numérotés de 0 à N-1, N étant la taille du BLOB. Voici un exemple :

```
      ` Déclarer une variable de type BLOB
C_BLOB (vBlob)
      ` Fixer la taille du BLOB à 256 octets
FIXER TAILLE BLOB (vBlob;256)
      ` La boucle suivante initialise les 256 octets du BLOB à zéro
```

```
Boucle (vOctet; 0;Taille BLOB (vBlob)-1)
    vBlob{vOctet}:=0
Fin de boucle
```

Comme vous pouvez adresser individuellement tous les octets d'un BLOB, vous pouvez littéralement stocker tout ce que vous voulez dans une variable ou un champ de type BLOB.

Routines 4D pour manipuler les BLOBs

4D fournit les routines suivantes pour travailler avec les BLOBs :

• FIXER TAILLE BLOB redimensionne la taille d'un champ ou d'une variable de type BLOB.
• Taille BLOB retourne la taille d'un champ ou d'une variable de type BLOB.
• DOCUMENT VERS BLOB et BLOB VERS DOCUMENT vous permettent de charger et d'écrire un document entier dans un champ ou une variable de type BLOB et inversement (en option, les data forks et les resource forks sur Macintosh).
• VARIABLE VERS BLOB et BLOB VERS VARIABLE, ainsi que LISTE VERS BLOB et BLOB vers liste vous permettent de stocker et de charger des variables 4D dans des BLOBs.
• COMPRESSER BLOB, DECOMPRESSER BLOB et LIRE PROPRIETES BLOB vous permettent de travailler avec des BLOBs compressés.
• Les commandes BLOB vers entier, BLOB vers entier long, BLOB vers reel, BLOB vers texte, ENTIER VERS BLOB, ENTIER LONG VERS BLOB, REEL VERS BLOB et TEXTE VERS BLOB vous permettent de manipuler et de structurer les données en provenance du disque, des ressources, du système d'exploitation, etc.
• SUPPRIMER DANS BLOB, INSERER DANS BLOB et COPIER BLOB permettent de gérer les gros volumes de données à l'intérieur des BLOBs.
• CRYPTER BLOB et DECRYPTER BLOB permettent de crypter et de décrypter des données dans vos bases 4D.

Ces commandes sont décrites dans ce chapitre. De plus, vous disposez des routines suivantes :

• C_BLOB déclare une variable de type BLOB (reportez-vous à la section Commandes du thème Compilateur).
• LIRE DONNEES CONTENEUR et AJOUTER DONNEES AU CONTENEUR vous permettent de gérer tout type de données stockées dans le presse-papiers (référez-vous à la section Gestion du conteneur de données).
• LIRE RESSOURCE et ECRIRE RESSOURCE vous permettent de gérer tout type de données stockées dans une ressource qui se trouve sur disque (référez-vous à la section Ressources).
• ENVOYER BLOB HTML vous permet d'envoyer tout type de données à un navigateur Web. Cette commande est incluse dans le thème Serveur Web.
• IMAGE VERS BLOB, BLOB VERS IMAGE et IMAGE VERS GIF vous permettent d'ouvrir et de convertir des images via des BLOBs. Ces commandes sont incluses dans le thème Images.
• GENERER CLES CRYPTAGE et GENERER DEMANDE CERTIFICAT permettent d'exploiter le protocole SSL (Secured Socket Layer) pour crypter des données ou les connexions Web. Ces commandes sont incluses dans le thème Protocole sécurisé.

BLOB VERS DOCUMENT (document; blob{; *})

Paramètre	Type		Description
document	Alpha	→	Nom du document
blob	BLOB	→	Nouveau contenu du document
*	*	→	Macintosh seulement : la resource fork est écrite si * est passé ; sinon la data fork est écrite

Description

BLOB VERS DOCUMENT écrit le contenu de document en utilisant les données stockées dans blob.
Vous pouvez passer dans document le nom d'un document existant ou non. Si le document n'existe pas, la commande le crée. Si vous passez le nom d'un document existant, assurez-vous qu'il n'est pas déjà ouvert, sinon une erreur est générée. Si vous voulez que l'utilisateur choisisse le document, appelez les routines Ouvrir document ou Creer document et utilisez la variable système Document (cf. exemple ci-dessous).

Notes pour les utilisateurs Mac OS :
• Les documents Macintosh peuvent être composés de deux éléments, la resource fork et la data fork ("partie des ressources" et "partie des données"). Par défaut, la commande BLOB VERS DOCUMENT réécrit la data fork du document. Si vous voulez réécrire la resource fork du document, passez le paramètre optionnel *. Sous Windows, le paramètre * est ignoré.
• Les documents générés par cette commande n'ont pas de "type". Si vous souhaitez créer un document avec type, vous devez utiliser la commande CHANGER TYPE DOCUMENT.

Exemple

Notre exemple est une base qui permet de stocker et de rechercher rapidement des documents. Dans un formulaire entrée, vous créez un bouton vous permettant de sauvegarder un document de votre choix qui contient des données provenant d'un champ de type BLOB. La méthode de ce bouton peut être la suivante :

```
$vhRefDoc:=Ouvrir document("") ` Sélectionner un document
Si (OK=1) ` Si un document a été choisi
    FERMER DOCUMENT($vhRefDoc) ` Nous ne voulons pas qu'il reste ouvert
    BLOB VERS DOCUMENT(Document;[VotreTable]VotreChampBLOB) ` Ecrire le contenu du document
    Si (OK=0)
        ` Gérer l'erreur
    Fin de si
Fin de si
```

Référence

Creer document, DOCUMENT VERS BLOB, Ouvrir document.

Variables système

La variable système OK prend la valeur à 1 si le document est correctement écrit. Sinon, elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

• Si vous essayez de réécrire un document qui est déjà ouvert par un autre process ou une autre application, une des Erreurs du gestionnaire de fichiers du système sera générée.

• L'espace sur disque peut être insuffisant pour l'écriture du contenu du document.

• Une erreur d'Entrée/Sortie peut survenir si le document est verrouillé, s'il se trouve sur un volume verrouillé ou si un problème survient lors de l'écriture du document.

Dans tous les cas, vous pouvez gérer les erreurs en utilisant la commande APPELER SUR ERREUR.

BLOB vers entier

version 6.0

BLOB vers entier (blob; ordreOctet{; offset}) → Numérique

Paramètre	Type		Description
blob	BLOB	→	BLOB duquel obtenir la valeur entière
ordreOctet	Numérique	→	0 Ordre d'octets mode natif
			1 Ordre d'octets Macintosh
			2 Ordre d'octets PC
offset	Variable	→	Offset (en octets) dans le BLOB
		←	Nouvel offset après la lecture
Résultat	Numérique	←	Valeur entière (2 octets)

Description

BLOB vers entier retourne une valeur entière (2 octets) lue dans le BLOB blob.

Le paramètre ordreOctet fixe l'ordre des octets ("byte ordering") de la valeur entière à lire. Vous pouvez passer une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets natif	Entier long	0
Ordre octets Macintosh	Entier long	1
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous ne passez pas de variable dans le paramètre optionnel offset, les deux premiers octets de BLOB sont lus.

Si vous passez une variable dans le paramètre optionnel offset, la valeur entière sur 2 octets est lue depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins 2. Sinon, une erreur -111 est générée.

Après l'exécution de la commande, la variable offset est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Exemple

L'exemple suivant lit 20 valeurs entières d'un BLOB à partir de l'offset 0x200 :

```
$vlOffset:=0x200
Boucle ($viBoucle;0;19)
    $viValeur:=BLOB vers entier(vxUnBlob;Ordre octets PC;$vlOffset)
        ` Faire quelque chose avec $viValeur
Fin de boucle
```

Référence

BLOB vers entier long, BLOB vers reel, BLOB vers texte, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

BLOB vers entier long

version 6.0

BLOB vers entier long (blob; ordreOctet{; offset}) → Numérique

Paramètre	Type		Description
blob	BLOB	→	BLOB duquel extraire la valeur de type Entier long
ordreOctet	Numérique	→	0 = Ordre d'octets natif 1 = Ordre d'octets Macintosh 2 = Ordre d'octets PC
offset	Variable	→	Offset (en octets) dans le BLOB
		←	Nouvel offset après lecture
Résultat	Numérique	←	Valeur de type Entier long (4 octets)

Description

La fonction BLOB vers entier long retourne une valeur de type Entier long (4 octets) lue dans le BLOB blob.

Le paramètre ordreOctet fixe l'ordre des octets ("byte ordering") de la valeur Entier long à lire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets natif	Entier long	0
Ordre octets Macintosh	Entier long	1
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous ne passez pas de variable dans le paramètre optionnel offset, les quatre premiers octets de BLOB sont lus.

Si vous passez une variable dans le paramètre optionnel offset, l'entier long sur 4 octets est lu depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins 4. Sinon, une erreur -111 est générée.

Après l'exécution de la commande, la variable offset est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Exemple

L'exemple suivant lit 20 valeurs de type Entier long dans un BLOB, à partir de l'offset 0x200 :

```
$vlOffset:=0x200
Boucle ($viBoucle;0;19)
   $vlValeur:=BLOB vers entier long(vxUnBlob;Ordre octets PC;$vlOffset)
      ` Faire quelque chose avec $vlValeur
Fin de boucle
```

Référence

BLOB vers entier, BLOB vers reel, BLOB vers texte, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

BLOB vers liste

version 6.0

BLOB vers liste (blob{; offset}) → RefList

Paramètre	Type		Description
blob	BLOB	→	BLOB contenant la liste hiérarchique
offset	Numérique	→	Offset (en octets) dans le BLOB
		←	Nouvel offset après la lecture
Résultat	RefList	←	Référence de la liste nouvellement créée

Description

BLOB vers liste crée une nouvelle liste hiérarchique avec les données stockées dans le BLOB blob à l'offset d'octet (à partir de zéro) spécifié par offset et retourne un numéro de référence de liste hiérarchique pour cette nouvelle liste.

Les données présentes dans le BLOB doivent être compatibles avec la commande : généralement, vous utilisez des BLOBs préalablement remplis avec la commande LISTE VERS BLOB.

Si vous ne passez pas le paramètre optionnel offset, les valeurs de la liste sont lues à partir du début du BLOB. Si vous gérez un BLOB dans lequel plusieurs variables ou listes ont été stockées, vous devez passer le paramètre offset ainsi qu'une variable numérique. Avant l'appel, fixez cette variable numérique à l'offset désiré. Après l'appel, cette même variable numérique retourne l'offset de la variable suivante stockée dans le BLOB.

Après l'appel, la variable OK prend la valeur 1 si la liste hiérarchique a été correctement créée. Si l'opération ne peut pas être effectuée à cause, par exemple, d'un manque de mémoire, la variable OK prend la valeur 0.

Note pour l'indépendance de plate-forme : BLOB vers liste et LISTE VERS BLOB utilisent un format interne 4D pour gérer les listes stockées dans des BLOBs. L'avantage est que vous n'avez pas besoin de vous soucier de la conversion des octets ("byte swapping") entre les plates-formes lorsque vous utilisez ces deux commandes. Autrement dit, avec ces commandes, un BLOB créé sous Windows peut être réutilisé sous Mac OS et vice-versa.

Exemple

Dans l'exemple suivant, la méthode d'un formulaire entrée extrait une liste d'un champ BLOB avant que le formulaire ne s'affiche puis le stocke dans le champ BLOB lorsque la saisie est validée :

```
` Méthode du formulaire [Choses à Faire];"Entrée"

Au cas ou

   : (Evenement formulaire=Sur chargement)
      hListe:=BLOB vers liste([Choses à Faire]Idées)
      Si (OK=0)
         hListe:=Nouvelle liste
      Fin de si

   : (Evenement formulaire=Sur libération)
      SUPPRIMER LISTE(hListe;*)

   : (bValider=1)
      LISTE VERS BLOB(hListe;[Choses à Faire]Idées)

Fin de cas
```

Référence

LISTE VERS BLOB.

Variables et ensembles système

La variable OK prend la valeur 1 si la liste a été correctement créée, sinon elle prend la valeur 0.

BLOB vers reel

version 6.0

BLOB vers reel (blob; formatRéel{; offset}) → Numérique

Paramètre	Type		Description
blob	BLOB	→	BLOB duquel extraire la valeur de type Réel
formatRéel	Numérique	→	0 Format réel natif
			1 Format réel étendu
			2 Format réel double Macintosh
			3 Format réel double Windows
offset	Variable	→	Offset (en octets) dans le BLOB
		←	Nouvel offset après lecture
Résultat	Numérique	←	Valeur de type Réel

Description

La fonction BLOB vers reel retourne une valeur de type Réel (ou Numérique) lue dans le BLOB blob.

Le paramètre formatRéel fixe le format interne et l'ordre des octets ("byte ordering") de la valeur de type Réel à lire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Format réel natif	Entier long	0
Format réel étendu	Entier long	1
Format réel double Macintosh	Entier long	2
Format réel double PC	Entier long	3

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous ne passez pas de variable dans le paramètre optionnel offset, les 8 ou 10 premiers octets de BLOB sont lus.

Si vous passez une variable dans le paramètre optionnel offset, la valeur réelle est lue depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins 8 ou 10. Sinon, une erreur -111 est générée.

Après l'exécution de la commande, la variable offset est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Exemple

L'exemple suivant lit 20 valeurs réelles dans un BLOB à partir de l'offset 0x200 :

```
$vlOffset:=0x200
Boucle ($viBoucle;0;19)
    $vrValeur:=BLOB vers reel(vxUnBlob;Format réel double PC;$vlOffset)
        ` Faire quelque chose avec $vrValeur
Fin de boucle
```

Référence

BLOB vers entier, BLOB vers entier long, BLOB vers texte, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

BLOB vers texte

version 11 (Modifiée)

BLOB vers texte (blob; formatTexte{; offset{; longueurTexte}}) → Chaîne

Paramètre	Type		Description
blob	BLOB	→	BLOB duquel extraire le texte
formatTexte	Numérique	→	Format et jeu de caractères du texte
offset	Variable	→	Offset (en octets) dans le BLOB
		←	Nouvel offset après la lecture
longueurTexte	Numérique	→	Nombre de caractères à lire
Résultat	Chaîne	←	Texte extrait

Description

La fonction BLOB vers texte retourne une valeur de type Texte lue dans le BLOB blob.

Le paramètre formatTexte définit le format interne et le jeu de caractères de la valeur de type Texte à lire. Dans les bases de données créées à compter de la version 11, 4D utilise par défaut le jeu de caractères Unicode (UTF8) pour la gestion des textes. Par compatibilité, cette commande permet de “forcer” l'utilisation du jeu de caractères Mac Roman (jeu de caractères utilisé dans les versions précédentes de 4D). Le choix du jeu de caractères s’effectue via le paramètre formatTexte. Pour cela, passez dans formatTexte une des constantes suivantes, placées dans le thème “BLOB” :

Constante	Type	Valeur
Mac Chaîne en C	Entier long	0
Mac Chaîne pascal	Entier long	1
Mac Texte avec longueur	Entier long	2
Mac Texte sans longueur	Entier long	3
UTF8 Chaîne en C	Entier long	4
UTF8 Texte avec longueur	Entier long	5
UTF8 Texte sans longueur	Entier long	6

Notes :
• Les constantes “UTF8” sont utilisables uniquement lorsque l’application fonctionne en mode Unicode.
• Les constantes “Mac” ne permettent pas de traiter des textes de plus de 32 ko.
• Si vous souhaitez manipuler des jeux de caractères autres que UTF8, utilisez la commande Convertir vers texte.

Pour plus d'informations sur ces constantes et les formats qu'elles représentent, reportez-vous à la description de la commande TEXTE VERS BLOB.

ATTENTION : Le nombre de caractères à lire est déterminé par le paramètre formatTexte, SAUF dans le cas des formats Mac Texte sans longueur et UTF8 Texte sans longueur pour lesquels vous devez spécifier le nombre de caractères à lire dans le paramètre longueurTexte. Pour les autres formats, longueurTexte est ignoré et vous pouvez l'omettre.

Si vous ne passez pas de variable dans le paramètre optionnel offset, les premiers octets de BLOB sont lus, en fonction de la valeur passée dans formatTexte. Notez que vous devez passer une variable dans le paramètre offset lorsque vous lisez une valeur de type Texte sans longueur.

Si vous passez une variable dans le paramètre optionnel offset, la valeur de type Texte est lue depuis l'offset exprimé en octets (à partir de zéro) du BLOB.

Note : Vous devez passer un offset compris entre 0 (zéro) et la taille du BLOB moins la taille du texte à extraire. Sinon, le résultat de la fonction ne sera pas exploitable.

Après l'exécution de la commande, la variable offset est incrémentée du nombre d'octets qui a été lu. Vous pouvez donc réutiliser la même variable avec une autre commande de lecture de BLOBs pour lire une autre valeur placée juste après celle que vous venez de lire.

Référence

BLOB vers entier, BLOB vers entier long, BLOB vers reel, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

BLOB VERS VARIABLE

version 6.0

BLOB VERS VARIABLE (blob; variable{; offset})

Paramètre	Type		Description
blob	BLOB	→	BLOB contenant une ou plusieurs variable(s) 4D
variable	Variable	←	Variable à écrire avec le contenu de BLOB
offset	Numérique	→	Position de la variable dans BLOB
		←	Position de la variable suivante dans BLOB

Description

BLOB VERS VARIABLE réécrit la variable variable avec les données stockées dans le BLOB blob à l'offset d'octet (à partir de zéro) spécifié par offset.

Les données dans le BLOB doivent être compatibles avec la variable de destination : vous utiliserez généralement des BLOBs que vous avez précédemment remplis à l'aide de VARIABLE VERS BLOB.

Si vous ne spécifiez pas le paramètre offset, les données de la variable sont lues à partir du début du BLOB. Si le BLOB contient plusieurs variables, vous devez passer le paramètre offset ainsi qu'une variable numérique. Avant d'appeler la commande, définissez cette variable numérique avec l'offset correspondant. Après l'appel, la même variable numérique retourne l'offset de la variable suivante stockée dans le BLOB.

La variable OK prend la valeur 1 si l'opération s'est correctement déroulée. Si l'opération n'a pas pu être effectuée, par exemple à cause d'un manque de mémoire, la variable OK prend la valeur 0.

Note sur l'indépendance de plate-forme : BLOB VERS VARIABLE et VARIABLE VERS BLOB utilisent un format interne à 4D pour gérer les variables stockées dans les BLOBs. Vous n'avez donc pas besoin de vous préoccuper de la conversion des octets ("byte swapping") entre les différentes plates-formes lors de l'utilisation de ces deux commandes. Un BLOB créé sous Windows à l'aide de ces deux commandes peut être réutilisé sans la moindre manipulation sous Mac OS et vice-versa.

Exemple

Référez-vous aux exemples de VARIABLE VERS BLOB.

Référence

VARIABLE VERS BLOB.

Variables et ensembles système

La variable OK prend la valeur 1 si la variable a été correctement réécrite, sinon elle prend la valeur 0.

COMPRESSER BLOB

version 6.5.3 (Modifiée)

COMPRESSER BLOB (blob{; compression})

Paramètre	Type		Description
blob	BLOB	→	BLOB à compresser
compression	Numérique	→	Si ce paramètre est passé : 1= taux de compression maximum 2 = vitesse de compression maximum

Description

COMPRESSER BLOB compresse le BLOB blob à l'aide de l'algorithme de compression interne de 4D.

Le paramètre optionnel compression vous permet de fixer la façon dont le BLOB sera compressé :
• Si vous passez 1, le BLOB est compressé de manière aussi compacte que possible, au détriment de la vitesse à laquelle la compression et la décompression sont effectuées.
• Si vous passez 2, le BLOB est compressé de manière aussi rapide que possible (et sera décompressé aussi vite que possible) au détriment du taux de compression (une fois compressé, le BLOB prend plus de place).
• Si vous passez une autre valeur ou si vous omettez ce paramètre, le BLOB est compressé de manière aussi compacte que possible (méthode de compression 1).

4D fournit les constantes suivantes:

Constante	Type	Valeur
Méthode de compression compacte	Entier long	1
Méthode de compression rapide	Entier long	2

Note : La commande compresse uniquement les BLOBs de taille supérieure ou égale à 255 octets.

Après que cette commande ait été appelée, la variable système OK prend la valeur 1 si le BLOB a été correctement compressé.
Si la compression n'a pu être effectuée, OK prend la valeur 0. Dans ce cas, si l'erreur provient du fait que la taille du BLOB est inférieure à 255 octets ou que la mémoire disponible est insuffisante pour effectuer l'opération, aucune erreur n'est générée, la méthode poursuit son exécution.

En revanche, si l'erreur est causée par un problème plus important (le BLOB est endommagé), l'erreur -10600 est générée. Cette erreur, relativement rare, peut être interceptée à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

Lorsqu'un BLOB a été compressé, vous pouvez le décompresser à l'aide de la commande DECOMPRESSER BLOB. Pour savoir si un BLOB a été compressé, utilisez la commande LIRE PROPRIETES BLOB.

ATTENTION : Un BLOB compressé est toujours un BLOB, rien ne vous empêche donc de modifier son contenu. Cependant, si vous le modifiez, la commande DECOMPRESSER BLOB ne pourra plus décompresser correctement le BLOB.

Exemples

(1) L'exemple suivant teste si le BLOB vxMonBlob est compressé et, sinon, le compresse :

```
LIRE PROPRIETES BLOB (vxBlob;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)
Si ($vlCompressé=Non compressé)
    COMPRESSER BLOB (vxMonBlob)
Fin de si
```

Notez que si vous appliquez COMPRESSER BLOB à un BLOB déjà compressé, la commande le détecte et ne fait rien.

(2) L'exemple suivant vous permet de sélectionner un document puis de le compresser :

```
$vhDocRef := Ouvrir document ("")
Si (OK=1)
    FERMER DOCUMENT ($vhDocRef)
    DOCUMENT VERS BLOB (Document;vxBlob)
    Si (OK=1)
        COMPRESSER BLOB (vxBlob)
        Si (OK=1)
            BLOB VERS DOCUMENT (Document;vxBlob)
        Fin de si
    Fin de si
Fin de si
```

Référence

DECOMPRESSER BLOB, LIRE PROPRIETES BLOB.

Variables et ensembles système

La variable OK prend la valeur 1 si le BLOB a été correctement compressé, sinon elle prend la valeur 0.

COPIER BLOB

version 6.0

COPIER BLOB (srcBLOB; dstBLOB; srcOffset; dstOffset; nombre)

Paramètre	Type		Description
srcBLOB	BLOB	→	BLOB source
dstBLOB	BLOB	→	BLOB de destination
srcOffset	Numérique	→	Position dans la source pour la copie
dstOffset	Numérique	→	Position dans la destination pour la copie
nombre	Numérique	→	Nombre d'octets à copier

Description

COPIER BLOB copie le nombre d'octets spécifié par nombre du BLOB srcBLOB vers le BLOB dstBLOB.

La copie commence à la position (exprimée par rapport à l'origine du BLOB source) définie par srcOffset et est placée à partir de la position (exprimée par rapport à l'origine du BLOB de destination) définie par dstOffset.

Notez que le BLOB de destination peut être redimensionné si nécessaire.

Référence

INSERER DANS BLOB, SUPPRIMER DANS BLOB.

CRYPTER BLOB

BLOB

version 6.7

CRYPTER BLOB (aCrypter; cléPrivEmetteur{; cléPubRécepteur})

Paramètre	Type		Description
aCrypter	BLOB	→	Données à crypter
		←	Données cryptées
cléPrivEmetteur	BLOB	→	Clé privée de l'émetteur
cléPubRécepteur	BLOB	→	Clé publique du récepteur

Description

La commande CRYPTER BLOB permet de crypter le contenu du BLOB aCrypter à l'aide de la clé privée de l'émetteur cléPrivEmetteur ainsi que, optionnellement, de la clé publique du récepteur cléPubRécepteur. Pour obtenir une paire de clés de cryptage (clé publique et clé privée), utilisez la routine GENERER CLES CRYPTAGE, placée dans le thème "Protocole sécurisé".

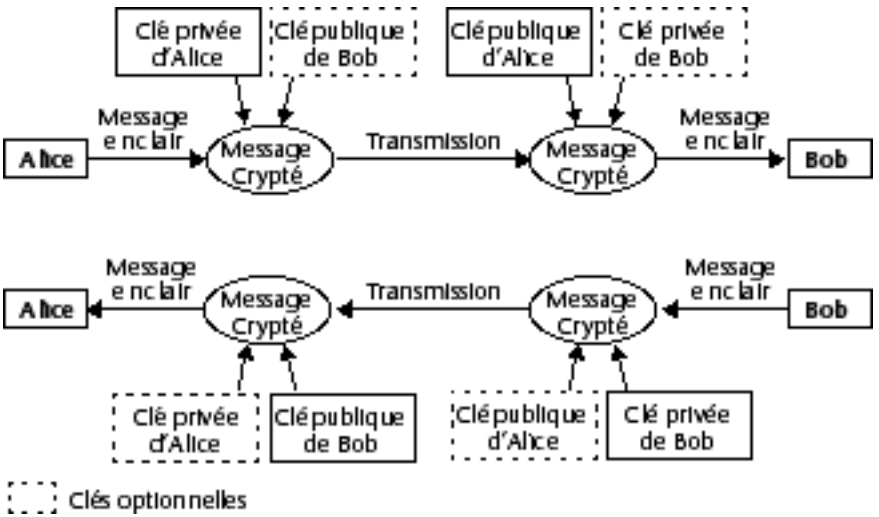
Note : La commande CRYPTER BLOB exploite l'algorithme et les fonctions de cryptage du protocole SSL. Par conséquent, pour pouvoir utiliser cette commande, vous devez veiller à ce que les composants nécessaires au fonctionnement du protocole SSL soient installés sur la machine — même si vous ne souhaitez pas utiliser SSL dans le cadre de connexions à un serveur Web 4D. Pour plus d'informations, reportez-vous à la section Services Web, Utiliser le protocole SSL.

• L'utilisation d'une seule clé pour le cryptage (clé privée de l'émetteur) garantit l'impossibilité pour toute personne ne disposant pas de la clé publique de lire les données. Elle garantit également que c'est bien l'émetteur qui a crypté les données.
• L'utilisation d'une paire de clés pour le cryptage (clé privée de l'émetteur + clé publique du récepteur) garantit en outre qu'un seul récepteur pourra lire les données.

Le format interne des BLOBs contenant des clés est le PKCS. Ce format standard, multi-plate-forme, permet l'échange ou la manipulation des clés par simple copier-coller dans un Email ou un fichier texte.

Après l'exécution de la commande, le BLOB aCrypter contient les données cryptées. Ces données ne pourront être décryptées qu'avec la commande DECRYPTER BLOB, à laquelle la clé publique de l'émetteur sera passée en paramètre. En outre, si la clé publique (optionnelle) du récepteur avait été utilisée pour le cryptage, la clé privée du récepteur sera également nécessaire pour le décryptage.

Principe du cryptage à clés publiques/privées pour l'échange de messages entre deux individus, "Alice" et "Bob" :



Note : L'algorithme de cryptage comporte une fonction de vérification d'intégrité (*checksum*), afin d'empêcher toute modification malveillante ou accidentelle du contenu du BLOB. Par conséquent, un BLOB crypté ne doit pas être modifié, sous peine de ne pas pouvoir être décrypté.

Optimisation des commandes de cryptage

Le cryptage des données ralentit l'exécution de l'application, en particulier si une paire de clés est utilisée. Deux types d'optimisations sont toutefois possibles :

• Suivant la quantité de mémoire disponible, la commande s'exécute en mode "synchrone" ou "asynchrone".
Le mode asynchrone est plus rapide, car il ne bloque pas les autres process. Ce mode est automatiquement utilisé si la mémoire disponible est au moins égale à 2 fois la taille de la source à crypter.
Dans le cas contraire, pour des raisons de sécurité, le mode synchrone est utilisé. Ce mode est plus lent car les autres process sont bloqués.

• Dans le cas de BLOBs volumineux, l'astuce consiste à crypter uniquement une partie déterminée et sensible du BLOB, afin de réduire la taille des données à traiter et donc le temps d'exécution.

Exemples

• Utilisation d'une seule clé
Une société veut garantir la confidentialité d'informations stockées dans une base 4D. Elle doit régulièrement envoyer ces données à ses filiales, par exemple sous la forme de fichiers via Internet.

1. La société commence par générer une paire de clés à l'aide de la commande GENERER CLES CRYPTAGE.

```
    `Méthode GENERE_CLES_TXT
C_BLOB($BcléPublique; $BcléPrivée)
GENERER CLES CRYPTAGE($BcléPrivée;$BcléPublique)
BLOB VERS DOCUMENT("cléPublique.txt"; $BcléPublique)
BLOB VERS DOCUMENT("cléPrivée.txt"; $BcléPrivée)
```

2. La société conserve la clé privée, et remet à chaque filiale une copie du document contenant la clé publique. Il faut, bien entendu, que cette transmission s'effectue d'une façon sûre, par exemple par la copie sur une disquette remise physiquement aux filiales.

3. Par la suite, la société copie les informations confidentielles (stockées par exemple dans un champ texte) dans des BLOBs et les crypte avec sa clé privée :

```
    `Méthode CRYPTER_INFOS
C_BLOB($vbCrypté;$vbcléPrivée)
C_TEXTE($vtCrypter)

$vtCrypter:=[Confidentiel]Info
VARIABLE VERS BLOB ($vtCrypter;$vbCrypté)
DOCUMENT VERS BLOB("cléPrivée.txt"; $vbcléPrivée)
Si (OK=1)
    CRYPTER BLOB ($vbCrypté; $vbcléPrivée)
    BLOB VERS DOCUMENT ("MiseAJour.txt";$vbCrypté)
Fin de si
```

4. Le fichier de mise à jour peut alors être envoyé aux filiales (même en passant par un canal non sécurisé comme Internet). Si un tiers intercepte le fichier crypté, il sera dans l'incapacité de le décrypter sans la clé publique.

5. Chaque filiale peut, quant à elle, décrypter le document à l'aide de la clé publique :

```
    `Méthode DECRYPTER_INFOS
C_BLOB($vbCrypté;$vbcléPublique)
C_TEXTE($vtDécrypté)
C_HEURE ($vhRefDoc)
```

```
ALERTE ("Veuillez sélectionner le document crypté.")
$vhRefDoc:=Ouvrir document("") `Sélection du fichier MiseAJour.txt
Si (OK=1)
    FERMER DOCUMENT($vhRefDoc)
    DOCUMENT VERS BLOB(Document;$vbCrypté)
    DOCUMENT VERS BLOB("cléPublique.txt"; $vbcléPublique)
    Si (OK=1)
        DECRYPTER BLOB ($vbCrypté; $vbcléPublique)
        BLOB VERS VARIABLE($vbCrypté; $vtDécrypté)
        CREER ENREGISTREMENT ([Confidentiel])
        [Confidentiel]Info:=$vtDécrypté
        STOCKER ENREGISTREMENT([Confidentiel])
    Fin de si
Fin de si
```

• Utilisation de deux clés

Une société souhaite utiliser un système d'échange de données via Internet dans lequel chaque filiale reçoit des informations confidentielles mais envoie également ses propres informations à la maison-mère. Ce système a donc les impératifs suivants :

- Seul le destinataire doit pouvoir lire un message,
- On doit avoir la garantie que le message provient bien de l'expéditeur.

1. La maison-mère ainsi que chaque filiale génèrent leurs propres paires de clés (à l'aide de la méthode *GENERE_CLES_TXT*).

```
    `Méthode GENERE_CLES_TXT
C_BLOB($BcléPublique; $BcléPrivée)
GENERER CLES CRYPTAGE($BcléPrivée;$BcléPublique)
BLOB VERS DOCUMENT("cléPublique.txt"; $BcléPublique)
BLOB VERS DOCUMENT("cléPrivée.txt"; $BcléPrivée)
```

2. Chacune garde sa clé privée. Chaque filiale envoie sa clé publique à la maison-mère, qui elle-même envoie sa clé publique à chaque filiale. Cette transmission ne doit pas nécessairement être effectuée par un canal protégé, car la seule détention de la clé publique dans ce cas sera insuffisante pour décrypter une information.

3. Pour crypter une information à envoyer, une filiale ou la maison-mère exécute la méthode *CRYPTER_INFOS_2* qui utilise la clé privée de l'émetteur et la clé publique du destinataire pour crypter les données :

```
    `Méthode CRYPTER_INFOS_2
C_BLOB($vbCrypté;$vbcléPrivée;$vbcléPublique)
C_TEXTE($vtCrypter)
C_HEURE ($vhRefDoc)
```

```
$vtCrypter:=[Confidentiel]Info
VARIABLE VERS BLOB ($vtCrypter;$vbCrypté)
    ` On charge sa propre clé privée...
DOCUMENT VERS BLOB("cléPrivée.txt"; $vbcléPrivée)
Si (OK=1)
        ` ...et la clé publique du récepteur
    ALERTE ("Veuillez sélectionner la clé publique du destinataire.")
    $vhRefDoc:=Ouvrir document("") `Sélection de la clé publique à charger
    Si (OK=1)
        FERMER DOCUMENT($vhRefDoc)
        DOCUMENT VERS BLOB(Document;$vbcléPublique)
            `Cryptage du BLOB avec les deux clés en paramètres
        CRYPTER BLOB ($vbCrypté; $vbcléPrivée; $vbcléPublique)
        BLOB VERS DOCUMENT ("MiseAJour.txt";$vbCrypté)
    Fin de si
Fin de si
```

4. Le fichier crypté peut alors être envoyé au destinataire via Internet. Si un tiers l'intercepte, il sera dans l'incapacité de le décrypter, même en connaissant les clés publiques, car il lui manquera la clé privée du destinataire.

5. Chaque destinataire peut, quant à lui, décrypter le document reçu, en utilisant sa clé privée et la clé publique de l'émetteur :

```
    `Méthode DECRYPTER_INFOS_2
C_BLOB($vbCrypté;$vbcléPublique;$vbcléPrivée)
C_TEXTE($vtDécrypté)
C_HEURE ($vhRefDoc)

ALERTE ("Veuillez sélectionner le document crypté.")
$vhRefDoc:=Ouvrir document("") `Sélection du fichier MiseAJour.txt
Si (OK=1)
    FERMER DOCUMENT($vhRefDoc)
    DOCUMENT VERS BLOB(Document;$vbCrypté)
        `On charge sa propre clé privée
    DOCUMENT VERS BLOB("cléPrivée.txt"; $vbcléPrivée)
    Si (OK=1)
        ` ...et la clé publique de l'émetteur
        ALERTE ("Veuillez sélectionner la clé publique de l'envoyeur.")
        $vhRefDoc:=Ouvrir document("") `Sélection de la clé publique
```

```
            Si (OK=1)
                FERMER DOCUMENT($vhRefDoc)
                DOCUMENT VERS BLOB(Document;$vbcléPublique)
                    `Décryptage du BLOB avec les deux clés en paramètres
                DECRYPTER BLOB ($vbCrypté; $vbcléPublique;$vbcléPrivée)
                BLOB VERS VARIABLE($vbCrypté; $vtDécrypté)
                CREER ENREGISTREMENT ([Confidentiel])
                [Confidentiel]Info:=$vtDécrypté
                STOCKER ENREGISTREMENT([Confidentiel])
            Fin de si
        Fin de si
    Fin de si
```

Référence

DECRYPTER BLOB, GENERER CLES CRYPTAGE.

version 6.5.3 (Modifiée)

DECOMPRESSER BLOB (blob)

Paramètre	Type		Description
blob	BLOB	→	BLOB à décompresser

Description

DECOMPRESSER BLOB décompresse le BLOB blob préalablement compressé à l'aide de la commande COMPRESSER BLOB.

Après l'appel de la commande, la variable système OK prend la valeur 1 si le BLOB a été correctement décompressé.

Si la décompression n'a pas pu être effectuée, OK prend la valeur 0.
Dans ce cas, si l'erreur provient du fait que la mémoire disponible est insuffisante pour effectuer l'opération, aucune erreur n'est générée et la méthode poursuit son exécution.
En revanche, si l'erreur est causée par un problème plus important (le BLOB n'avait pas été compressé ou le BLOB est endommagé), l'erreur -10600 est générée. Cette erreur peut être interceptée à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

De manière générale, il est préférable d'appeler la commande LIRE PROPRIETES BLOB pour savoir si le BLOB a été compressé avant d'exécuter DECOMPRESSER BLOB.

Exemples

(1) L'exemple suivant teste si le BLOB vxMonBlob est compressé et, si oui, le décompresse :

```
LIRE PROPRIETES BLOB (vxMonBlob;$vlCompressé;$vlTailleDécompressée;
                                                    $vlTailleCourante)
Si ($vlCompressé#Non compressé)
   DECOMPRESSER BLOB (vxMonBlob)
Fin de si
```

(2) L'exemple suivant vous permet de sélectionner un document et puis de le décompresser s'il était compressé :

```
$vhDocRef := Ouvrir document ("")
Si (OK=1)
    FERMER DOCUMENT ($vhDocRef)
    DOCUMENT VERS BLOB (Document;vxBlob)
    Si (OK=1)
        LIRE PROPRIETES BLOB (vxBlob;$vlCompressé;$vlTailleDécompressée;
                                                                $vlTailleCourante)
        Si ($vlCompressé#Non compressé)
            DECOMPRESSER BLOB (vxBlob)
            Si (OK=1)
                BLOB VERS DOCUMENT (Document;vxBlob)
            Fin de si
        Fin de si
    Fin de si
Fin de si
```

Référence

COMPRESSER BLOB, LIRE PROPRIETES BLOB.

Variables et ensembles système

La variable OK prend la valeur 1 si le BLOB a été correctement décompressé, sinon elle prend la valeur 0.

DECRYPTER BLOB

version 6.7

DECRYPTER BLOB (aDécrypter; cléPubEmetteur{; cléPrivRécepteur})

Paramètre	Type		Description
aDécrypter	BLOB	→	Données à décrypter
		←	Données décryptées
cléPubEmetteur	BLOB	→	Clé publique de l'émetteur
cléPrivRécepteur	BLOB	→	Clé privée du récepteur

Description

La commande DECRYPTER BLOB permet de décrypter le contenu du BLOB aDécrypter à l'aide de la clé publique de l'émetteur cléPubEmetteur ainsi que, optionnellement, de la clé privée du récepteur cléPrivRécepteur.

Vous passez dans le paramètre cléPubEmetteur le BLOB contenant la clé publique de l'émetteur. Cette clé a été générée par l'émetteur (à l'aide de la commande GENERER CLES CRYPTAGE), qu'il doit ensuite transmettre au récepteur.

Le paramètre optionnel cléPrivRécepteur doit recevoir la clé privée du récepteur. Dans ce cas, le récepteur doit également avoir généré une paire de clés de cryptage à l'aide de GENERER CLES CRYPTAGE et transmis sa clé publique à l'émetteur. Le système de cryptage à deux clés permet de garantir que seul l'émetteur peut avoir crypté le message et seul le récepteur peut le décrypter.
Pour plus d'informations sur le système de cryptage à deux clés, reportez-vous à la description de la commande CRYPTER BLOB.

La commande DECRYPTER BLOB comporte une fonction de vérification d'intégrité (*checksum*), afin d'empêcher toute modification malveillante ou accidentelle du contenu du BLOB. Si le BLOB crypté est endommagé ou modifié, la commande ne fera rien et retournera une erreur.

Exemple

Reportez-vous aux exemples de la commande CRYPTER BLOB.

Référence

CRYPTER BLOB, GENERER CLES CRYPTAGE.

DOCUMENT VERS BLOB (document; blob{; *})

Paramètre	Type		Description
document	Alpha	→	Nom du document
blob	BLOB	→	Champ ou variable de type BLOB devant recevoir le document
		←	Contenu du document
*		→	Macintosh seulement : la resource fork est chargée si * est passé ; sinon, la data fork est chargée

Description

DOCUMENT VERS BLOB charge le contenu de document dans blob. Vous devez passer un nom de document valide, c'est-à-dire qui désigne un document existant qui n'est pas déjà ouvert, sinon une erreur sera générée. Si vous voulez que l'utilisateur choisisse le document, utilisez la routine Ouvrir document et la variable système Document (cf. l'exemple ci-dessous).

Note pour les utilisateurs Mac OS : les documents Macintosh peuvent être composés de deux éléments, la resource fork et la data fork ("partie des ressources" et "partie des données"). Par défaut, la commande DOCUMENT VERS BLOB charge la data fork du document. Si vous voulez charger la resource fork du document, passez le paramètre optionnel *.
Sous Windows, le paramètre * est ignoré. Notez que l'environnement 4D vous fournit l'équivalent des resource forks de Mac OS sous Windows : par exemple, la data fork d'une base 4D est stockée dans un fichier comportant l'extension .4DB et la resource fork est stockée dans un fichier du même nom, mais comportant l'extension .RSR. Si vous développez une application 4D qui gère les data forks et les resource forks stockées dans des BLOBs, sous Windows, il vous suffit d'accéder au fichier correspondant à la "fork" avec laquelle vous voulez travailler.

Exemple

Notre exemple est une base qui vous permet de stocker et chercher rapidement des documents. Dans un formulaire entrée, vous créez un bouton qui vous permet de charger un document de votre choix dans un champ de type BLOB. La méthode pour ce bouton peut être la suivante :

```
$vhRefDoc:=Ouvrir document("") ` Sélectionner un document
Si (OK=1) ` Si un document a été choisi
    FERMER DOCUMENT($vhRefDoc) ` Nous voulons pas qu'il reste ouvert
        ` Charger le document
    DOCUMENT VERS BLOB (Document;[VotreTable]VotreChampBLOB)
    Si (OK=0)
        ` Gérer l'erreur
    Fin de si
Fin de si
```

Référence

BLOB VERS DOCUMENT, Ouvrir document.

Variables système

La variable système OK prend la valeur 1 si le document est correctement lu. Sinon, elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

• Si vous essayez de charger dans un BLOB un document qui n'existe pas ou qui est déjà ouvert par un(e) autre process ou application, une des Erreurs du gestionnaire de fichiers du système sera générée.

• Une erreur d'Entrée/Sortie peut survenir si le document est verrouillé, s'il se trouve sur un volume verrouillé ou si un problème survient pendant la lecture du document.

• S'il n'y a pas assez de mémoire pour charger le document, une erreur -108 est générée.

Dans tous les cas, vous pouvez gérer les erreurs en utilisant la commande APPELER SUR ERREUR.

ENTIER LONG VERS BLOB

version 6.0

ENTIER LONG VERS BLOB (entierLong; blob; ordreOctets{; offset | *})

Paramètre	Type		Description
entierLong	Numérique	→	Valeur de type Entier long à écrire dans BLOB
blob	BLOB	→	BLOB devant recevoir l'entier long
ordreOctets	Numérique	→	0 Ordre d'octets natif
			1 Ordre d'octets Macintosh
			2 Ordre d'octets PC
offset | *	Variable | *	→	Offset (en octets) dans le BLOB
			ou * pour ajouter la valeur à la fin du BLOB
		←	Nouvel offset après l'écriture si * omis

Description

La commande ENTIER LONG VERS BLOB écrit la valeur de type Entier long (4 octets) entierLong dans le BLOB blob.

Le paramètre ordreOctet fixe l'ordre des octets ("byte ordering") de la valeur Entier long à écrire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets natif	Entier long	0
Ordre octets Macintosh	Entier long	1
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous passez le paramètre optionnel *, l'entier long sur 4 octets est ajouté à la fin du BLOB et la taille de blob est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre offset, l'entier long est stocké au début de blob en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre offset, l'entier long est écrit à partir de l'offset offset, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez l'entier long, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à 4 octets le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre offset est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemples

(1) Après l'exécution de ce code :

ENTIER LONG VERS BLOB (0x01020304;vxBlob;Ordre octets natif)

- La taille de vxBlob est 4 octets
- Sur plate-forme PowerPC vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04
- Sur plate-forme Intel vxBLOB{0}=$04, vxBLOB{1}=$03, vxBLOB{2}=$02, vxBLOB{3}=$01

(2) Après l'exécution de ce code :

ENTIER LONG VERS BLOB (0x01020304;vxBlob;Ordre octets Macintosh)

- La taille de vxBlob est 4 octets
- Sur toutes les plates-formes vxBLOB{0}=$01, vxBLOB{1}=$02, vxBLOB{2}=$03, vxBLOB{3}=$04

(3) Après l'exécution de ce code :

ENTIER LONG VERS BLOB (0x01020304;vxBlob;Ordre octets PC)

- La taille de vxBlob est 4 octets
- Sur toutes les plates-formes vxBLOB{0}=$04, vxBLOB{1}=$03, vxBLOB{2}=$02, vxBLOB{3}=$01

(4) Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;100)
ENTIER LONG VERS BLOB (0x01020304;vxBlob;Ordre octets PC)
```

• La taille de vxBlob est 104 octets
• Sur toutes les plates-formes vxBLOB{100}=$04, vxBLOB{101}=$03, vxBLOB{102}=$02, vxBLOB{103}=$01
• Les autres octets du BLOB sont inchangés

(5) Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;100)
vlOffset:=50
ENTIER LONG VERS BLOB (0x01020304;vxBlob;Ordre octets Macintosh;vlOffset)
```

• La taille de vxBlob est 100 K
• Sur toutes les plates-formes vxBLOB{50}=$01, vxBLOB{51}=$02, vxBLOB{52}=$03, vxBLOB{53}=$04
• Les autres octets du BLOB restent inchangés
• La variable vlOffset a été incrémentée de 4 (et est alors égale à 54)

Référence

BLOB vers entier, BLOB vers entier long, BLOB vers reel, BLOB vers texte, ENTIER VERS BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

ENTIER VERS BLOB

version 6.0

ENTIER VERS BLOB (entier; blob; ordreOctet{; offset | *})

Paramètre	Type		Description
entier	Numérique	→	Valeur entière à écrire dans le BLOB
blob	BLOB	→	BLOB devant recevoir la valeur entière
ordreOctet	Numérique	→	0 Ordre des octets en mode natif
			1 Ordre des octets Macintosh
			2 Ordre des octets PC
offset | *	Variable | *	→	Offset (en octets) de l'entier dans le BLOB
			ou * pour ajouter la valeur à la fin du BLOB
		←	Nouvel offset après écriture si * omis

Description

ENTIER VERS BLOB écrit la valeur entière (2 octets) entier dans le BLOB blob.

Le paramètre ordreOctet fixe l'ordre des octets ("byte ordering") de la valeur entière à écrire. Vous pouvez passer une des constantes fournies par 4D :

Constante	Type	Valeur
Ordre octets natif	Entier long	0
Ordre octets Macintosh	Entier long	1
Ordre octets PC	Entier long	2

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette commande.

Si vous passez le paramètre optionnel *, la valeur entière sur 2 octets est ajoutée à la fin du BLOB et sa taille est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre offset, la valeur entière est stockée au début de blob en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre offset, la valeur entière est écrite à partir de l'offset offset, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez l'entier, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à 2 octets le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre offset est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemples

(1) Après l'exécution de ce code :

```
ENTIER VERS BLOB (0x0206;vxBlob;Ordre octets natif)
```

- La taille de vxBlob est 2 octets
- Sur plate-forme PowerPC vxBLOB{0} = $02 et vxBLOB{1} = $06
- Sur plate-forme Intel vxBLOB{0} = $06 et vxBLOB{1} = $02

(2) Après l'exécution de ce code :

```
ENTIER VERS BLOB (0x0206;vxBlob;Ordre octets Macintosh)
```

- La taille de vxBlob est 2 octets
- Sur toutes les plates-formes vxBLOB{0} = $02 et vxBLOB{1} = $06

(3) Après l'exécution de ce code :

```
ENTIER VERS BLOB (0x0206;vxBlob;Ordre octets PC)
```

- La taille de vxBlob est 2 octets
- Sur toutes les plates-formes vxBLOB{0} = $06 et vxBLOB{1} = $02

(4) Après l'exécution de ce code:

```
FIXER TAILLE BLOB (vxBlob;100)
ENTIER VERS BLOB (0x0206;vxBlob;Ordre octets PC;*)
```

- La taille de vxBlob est 102 octets
- Sur toutes les plates-formes vxBLOB{100} = $06 et vxBLOB{101} = $02
- Les autres octets du BLOB restent inchangés

(5) Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;100)
vlOffset:=50
ENTIER VERS BLOB (518;vxBlob;Ordre octets Macintosh;vlOffset)
```

- La taille de vxBlob est 100 octets
- Sur toutes les plates-formes vxBLOB{50} = $02 et vxBLOB{51} = $06
- Les autres octets du BLOB restent inchangés
- La variable vlOffset est incrémentée de 2 (et est alors égale à 52)

Référence

BLOB vers entier, BLOB vers entier long, BLOB vers reel, BLOB vers texte, ENTIER LONG VERS BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

FIXER TAILLE BLOB

version 6.0

FIXER TAILLE BLOB (blob; taille{; remplisseur})

Paramètre	Type		Description
blob	BLOB	→	Champ ou variable de type BLOB
taille	Numérique	→	Nouvelle taille de BLOB
remplisseur	Numérique	→	Code ASCII du caractère de remplissage

Description

FIXER TAILLE BLOB redimensionne blob selon la valeur passée dans le paramètre taille.

Si vous souhaitez que les nouveaux octets réservés (s'il y en a) pour le BLOB soient initialisés avec une valeur particulière, passez cette valeur (comprise entre 0 et 255) dans le paramètre optionnel remplisseur.

Exemples

(1) Lorsque vous n'avez plus besoin d'un BLOB process ou interprocess, il est préférable de libérer la mémoire qu'il occupe. Pour cela, écrivez le code suivant :

```
FIXER TAILLE BLOB(vProcessBLOB;0)
FIXER TAILLE BLOB(<>vInterprocessBLOB;0)
```

(2) L'exemple suivant crée un BLOB de 16 Ko et remplit chaque octet avec la valeur 0xFF :

```
C_BLOB(vxData)
FIXER TAILLE BLOB(vxData;16*1024;0xFF)
```

Référence

Taille BLOB.

Gestion des erreurs

Si vous ne pouvez pas redimensionner le BLOB parce qu'il n'y a pas assez de mémoire, l'erreur –108 est générée. Vous pouvez installer une méthode avec la commande APPELER SUR ERREUR pour interrompre la méthode lorsqu'une erreur survient.

INSERER DANS BLOB

version 6.0

INSERER DANS BLOB (blob; offset; nombre{; défaut})

Paramètre	Type		Description
blob	BLOB	→	BLOB dans lequel insérer les octets
offset	Numérique	→	Position de début d'insertion des octets
nombre	Numérique	→	Nombre d'octets à insérer
défaut	Numérique	→	Valeur d'octet par défaut (0x00..0xFF) 0x00 si ce paramètre est omis

Description

INSERER DANS BLOB insère le nombre d'octets spécifié par nombre dans le BLOB blob à la position spécifiée par offset. La taille du BLOB est augmentée de nombre d'octets.

Si vous ne passez pas le paramètre optionnel défaut, la valeur des octets insérés dans le BLOB est fixée à 0x00. Sinon, les octets prennent la valeur passée dans défaut (modulo 256 — 0..255).

Vous passez dans le paramètre offset la position (relative à l'origine du BLOB) de l'insertion.

Référence

SUPPRIMER DANS BLOB.

LIRE PROPRIETES BLOB

version 6.0

LIRE PROPRIETES BLOB (blob; compressé{; tailleDécompressée{; tailleCourante}})

Paramètre	Type		Description
blob	BLOB	→	BLOB sur lequel vous voulez obtenir des informations
compressé	Numérique	←	0 = BLOB n'est pas compressé 1 = BLOB compressé méthode compacte 2 = BLOB compressé méthode rapide
tailleDécompressée	Numérique	←	Taille du BLOB décompressé en octets
tailleCourante	Numérique	←	Taille courante du BLOB en octets

Description

LIRE PROPRIETES BLOB retourne des informations sur le BLOB blob.

• Le paramètre compressé vous indique si le BLOB est compressé ou non et retourne une des valeurs suivantes :

Constante	Type	Valeur
Non compressé	Entier long	0
Méthode de compression compacte	Entier long	1
Méthode de compression rapide	Entier long	2

Note : Les constantes ci-dessus sont fournies par 4D.

• Quel que soit l'état de compression du BLOB, le paramètre tailleDécompressée retourne la taille du BLOB non compressé.
• Le paramètre tailleCourante retourne la taille courante du BLOB. Si le BLOB est compressé, tailleCourante sera inférieur à tailleDécompressée. Si le BLOB n'est pas compressé, tailleCourante sera égal à tailleDécompressée.

Exemples

(1) Référez-vous aux exemples des commandes COMPRESSER BLOB et DECOMPRESSER BLOB.

(2) Lorsqu'un BLOB est compressé, la méthode projet suivante vous permet de connaître le taux de place gagnée en compressant le BLOB :

```
` Méthode projet Place gagnée par compression
` Place gagnée par compression (Pointeur {; Pointeur } ) -> Entier long
` Place gagnée par compression ( -> BLOB {; -> octetsGagnés } ) -> Pourcentage

C_POINTEUR ($1;$2)
C_ENTIER LONG ($0;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)

LIRE PROPRIETES BLOB ($1->;$vlCompressé;$vlTailleDécompressée;$vlTailleCourante)
Si ($vlTailleDécompressée=0)
   $0:=0
   Si (Nombre de parametres>=2)
      $2->:=0
   Fin de si
Sinon
   $0:=100-(($vlTailleCourante/$vlTailleDécompressée)*100)
   Si (Nombre de parametres>=2)
      $2->:=$vlTailleDécompressée-$vlTailleCourante
   Fin de si
Fin de si
```

Lorsque cette méthode est placée dans votre application, vous pouvez écrire :

```
` ...
COMPRESSER BLOB (vxBlob)
$vlPourcent:=Place gagnée par compression  (->vxBlob;->vlTailleBlob)
ALERTE ("La compression permet de gagner "+Chaine (vlTailleBlob)+" octets, donc "+
                                                Chaine($vlPourcent;"#0%")+" d'espace.")
```

Référence

COMPRESSER BLOB, DECOMPRESSER BLOB.

LISTE VERS BLOB

version 6.0

LISTE VERS BLOB (liste; blob{; *})

Paramètre	Type		Description
liste	RefList	→	Liste hiérarchique à stocker dans le BLOB
blob	BLOB	→	BLOB devant recevoir la liste hiérarchique
*	*	→	Ajouter la liste à la fin du BLOB

Description

La commande LISTE VERS BLOB stocke la liste hiérarchique liste dans le BLOB blob.

Si vous passez le paramètre optionnel *, la liste hiérarchique est ajoutée à la fin du BLOB et la taille de blob est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de variables ou de listes (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel *, la liste hiérarchique est stockée au début de blob en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Quel que soit l'endroit où vous placez la liste, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à la taille de la liste le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

ATTENTION : Si vous utilisez un BLOB pour stocker des listes, appelez ensuite la commande BLOB vers liste pour relire le contenu du BLOB car les listes sont stockées dans les BLOBs avec un format interne 4D.

Après l'exécution de la commande, la variable OK prend la valeur 1 si la liste hiérarchique a été correctement stockée. Si l'opération n'a pas pu être effectuée car, par exemple, il n'y avait pas assez de mémoire disponible, la variable OK prend la valeur 0.

Note pour l'indépendance de plate-forme : LISTE VERS BLOB et BLOB vers liste utilisent un format interne 4D pour gérer les listes stockées dans des BLOBs. L'avantage est que vous n'avez pas besoin de vous soucier de la conversion des octets ("byte swapping") entre les plates-formes lorsque vous utilisez ces deux commandes. Autrement dit, avec ces commandes, un BLOB créé sous Windows peut être réutilisé sous Mac OS et vice-versa.

Exemple

Reportez-vous à l'exemple de la fonction BLOB vers liste.

Référence

BLOB vers liste, BLOB VERS VARIABLE, STOCKER LISTE, VARIABLE VERS BLOB.

Variables et ensembles système

La variable OK prend la valeur 1 si la liste a été correctement écrite, sinon elle prend la valeur 0.

REEL VERS BLOB (réel; blob; formatRéel{; offset | *})

Paramètre	Type		Description
réel	Numérique	→	Valeur de type Réel à écrire dans le BLOB
blob	BLOB	→	BLOB devant recevoir la valeur Réel
formatRéel	Numérique	→	0 Format réel natif
			1 Format réel étendu
			2 Format réel double Macintosh
			3 Format réel double Windows
offset | *	Variable | *	→	Offset (en octets) dans le BLOB ou
			* pour ajouter la valeur à la fin du BLOB
		←	Nouvel offset après l'écriture si * omis

Description

La commande REEL VERS BLOB écrit la valeur de type Réel (ou Numérique) réel dans le BLOB blob.

Le paramètre formatRéel fixe le format interne et l'ordre des octets ("byte ordering") de la valeur de type Réel à écrire. Vous passez une des constantes fournies par 4D :

Constante	Type	Valeur
Format réel natif	Entier long	0
Format réel étendu	Entier long	1
Format réel double Macintosh	Entier long	2
Format réel double PC	Entier long	3

Note sur l'indépendance de plate-forme : Si vous échangez des BLOBs entre les plates-formes Macintosh et PC, il vous incombe de traiter les conversions d'octets ("byte swapping") lorsque vous utilisez cette fonction.

Si vous passez le paramètre optionnel *, la valeur réelle est ajoutée à la fin du BLOB et la taille de blob est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre offset, la valeur réelle est stockée au début de blob en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre offset, le réel est écrit à partir de l'offset offset, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez la valeur, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à 8 ou 10 octets le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre offset est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemples

(1) Après l'exécution de ce code :

```
C_REEL (vrValeur)
vrValeur := ...
REEL VERS BLOB (vrValeur;vxBlob;Format réel natif)
```

• Sur toutes les plates-formes, la taille de vxBlob est 8 octets

(2) Après l'exécution de ce code :

```
C_REEL (vrValeur)
vrValeur := ...
REEL VERS BLOB (vrValeur;vxBlob;Format réel étendu)
```

• Sur toutes les plates-formes, la taille de vxBlob est 10 octets

(3) Après l'exécution de ce code :

```
C_REEL (vrValeur)
vrValeur := ...
REEL VERS BLOB (vrValeur;vxBlob;Format réel double Macintosh) ` ou Format réel double PC
```

• Sur toutes les plates-formes, la taille de vxBlob est 8 octets

(4) Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;100)
C_REEL (vrValeur)
vrValeur := ...
REEL VERS BLOB (vrValeur;vxBlob;Format réel double PC)
    ` ou Format réel double Macintosh
```

• Sur toutes les plates-formes, la taille de vxBlob est 8 octets

(5) Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;100)
REEL VERS BLOB (vrValeur;vxBlob;Format réel étendu;*)
```

• Sur toutes les plates-formes, la taille de vxBlob est 110 octets
• Sur toutes les plates-formes, la valeur numérique est stockée dans les octets #100 à #109
• Les autres octets du BLOB restent inchangés

(6) Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;100)
C_REEL (vrValeur)
vrValeur := ...
vlOffset:=50
REEL VERS BLOB (vrValeur;vxBlob;Format réel double PC;vlOffset)
    ` ou Format réel double Macintosh
```

• Sur toutes les plates-formes, la taille de vxBlob est 100 octets
• Sur toutes les plates-formes, la valeur numérique est stockée dans les octets #50 à #57
• Les autres octets du BLOB restent inchangés
• La variable vlOffset est incrémentée de 8 (et est alors égale à 58)

Référence

BLOB vers entier, BLOB vers entier long, BLOB vers reel, BLOB vers texte, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, TEXTE VERS BLOB.

SUPPRIMER DANS BLOB

version 6.0

SUPPRIMER DANS BLOB (blob; offset; nombre)

Paramètre	Type		Description
blob	BLOB	→	BLOB duquel supprimer des octets
offset	Numérique	→	Offset à partir duquel supprimer les octets
nombre	Numérique	→	Nombre d'octets à supprimer

Description

SUPPRIMER DANS BLOB supprime le nombre d'octets spécifié par nombre du BLOB blob à partir de la position définie par offset (exprimée de manière relative à l'origine du BLOB). La taille du BLOB est réduite de nombre d'octets.

Référence

INSERER DANS BLOB.

Taille BLOB

version 6.0

Taille BLOB (blob) → Entier long

Paramètre	Type		Description
blob	BLOB	→	Champ ou variable de type BLOB
Résultat	Entier long	←	Taille en octets du BLOB

Description

Taille BLOB retourne la taille de blob exprimée en octets.

Exemple

La ligne de code suivante ajoute 100 octets au BLOB monBlob :

```
FIXER TAILLE BLOB (Taille BLOB(monBlob)+100)
```

Référence

FIXER TAILLE BLOB.

TEXTE VERS BLOB

BLOB

version 11 (Modifiée)

TEXTE VERS BLOB (texte; blob{; formatTexte{; offset | *}})

Paramètre	Type		Description
texte	Chaîne	→	Texte à écrire dans blob
blob	BLOB	→	BLOB devant recevoir le texte
formatTexte	Numérique	→	Format et jeu de caractères du texte
offset | *	Variable | *	→	Offset (en octets) dans le BLOB ou * pour ajouter la valeur à la fin du BLOB
		←	Nouvel offset après l'écriture si * omis

Description

La commande TEXTE VERS BLOB écrit la valeur de type Texte texte dans le BLOB blob.

Le paramètre formatTexte permet de définir le format interne et le jeu de caractères de la valeur de type Texte à écrire. Pour cela, passez dans formatTexte une des constantes suivantes, placées dans le thème “BLOB” :

Constante	Type	Valeur
Mac Chaîne en C	Entier long	0
Mac Chaîne pascal	Entier long	1
Mac Texte avec longueur	Entier long	2
Mac Texte sans longueur	Entier long	3
UTF8 Chaîne en C	Entier long	4
UTF8 Texte avec longueur	Entier long	5
UTF8 Texte sans longueur	Entier long	6

Si vous omettez le paramètre formatTexte, par défaut 4D utilise le format Mac Chaîne en C. Dans les bases de données créées à compter de la version 11, 4D travaille par défaut avec le jeu de caractères Unicode (UTF8) pour la gestion des textes, il est donc recommandé d'utiliser ce jeu de caractères.

Notes :

• Les constantes “UTF8” sont utilisables uniquement lorsque l’application fonctionne en mode Unicode.
• Les constantes “Mac” ne permettent pas de traiter des textes de plus de 32 ko.
• Si vous souhaitez manipuler des jeux de caractères autres que UTF8, utilisez la commande CONVERTIR DEPUIS TEXTE.

Le tableau suivant décrit chacun de ces formats :

Format texte	Description et Exemples
Chaîne en C	Le texte se termine par un caractère NULL (code ASCII $00).
UTF8	"" → $00
	"Café" → $43 61 66 C3 A9 00
Mac	"" → $00
	"Café" → $43 61 66 8E 00
Chaîne pascal	Le texte est précédé d'un octet de longueur.
UTF8	-
	-
Mac	"" → $00
	"Café" → $04 43 61 66 8E
Texte avec longueur	Le texte est précédé de 3 octets (UTF8) ou 2 octets (Mac) de longueur.
UTF8	"" → $00 00 00 00
	"Café" → $00 00 00 05 43 61 66 C3 A9
Mac	"" → $00 00
	"Café" → $00 04 43 61 66 8E
Texte sans longueur	Le texte est composé seulement de ses caractères.
UTF8	"" → Pas de valeur
	"Café" → $43 61 66 C3 A9
Mac	"" → Pas de valeur
	"Café" → $43 61 66 8E

Si vous passez le paramètre optionnel *, la valeur de type Texte est ajoutée à la fin du BLOB et la taille de blob est modifiée en conséquence. Ainsi, à l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de valeurs de type Entier, Entier long, Numérique ou Texte (référez-vous aux autres commandes sur les BLOBs) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre offset, la valeur de type Texte est stockée au début de blob en remplaçant son contenu précédent, et la taille du BLOB est modifiée en conséquence.

Si vous passez une variable dans le paramètre offset, la valeur de type Texte est écrite à l'offset offset, exprimé en octets (à partir de zéro), du BLOB. Quel que soit l'endroit où vous placez la valeur, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (plus jusqu'à la taille du texte le cas échéant). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre offset est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre valeur juste après celle que vous venez d'écrire.

Exemple

Après l'exécution de ce code :

```
FIXER TAILLE BLOB (vxBlob;0)
C_TEXTE (vtValeur)
vtValeur := "Café"  ` La longueur de vtValeur est de 4 octets
TEXTE VERS BLOB (vtValeur;vxBlob;Mac Chaîne en C)  ` La taille du BLOB devient 5 octets
TEXTE VERS BLOB (vtValeur;vxBlob;Mac Chaîne pascal)
    ` La taille du BLOB devient 5 octets
TEXTE VERS BLOB (vtValeur;vxBlob;Mac Texte avec longueur)
    ` La taille du BLOB devient 6 octets
TEXTE VERS BLOB (vtValeur;vxBlob;Mac Texte sans longueur)
    ` La taille du BLOB devient 4 octets
TEXTE VERS BLOB (vtValeur;vxBlob;UTF8 Chaîne en C)  ` La taille du BLOB devient 6 octets
TEXTE VERS BLOB (vtValeur;vxBlob;UTF8 Texte avec longueur)
    ` La taille du BLOB devient 9 octets
TEXTE VERS BLOB (vtValeur;vxBlob;UTF8 Texte sans longueur)
    ` La taille du BLOB devient 5 octets
```

Référence

BLOB vers entier, BLOB vers entier long, BLOB vers reel, BLOB vers texte, CONVERTIR DEPUIS TEXTE, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, REEL VERS BLOB.

VARIABLE VERS BLOB

version 6.0

VARIABLE VERS BLOB (variable; blob{; offset | *})

Paramètre	Type		Description
variable	Variable	→	Variable à stocker dans le BLOB
blob	BLOB	→	BLOB devant recevoir la variable
offset | *	Variable | *	→	Offset de la variable (en octets) dans BLOB ou * pour ajouter la variable à la fin du BLOB
		←	Nouvel offset après écriture si * omis

Description

VARIABLE VERS BLOB stocke la variable variable dans le BLOB blob.

Si vous passez le paramètre optionnel *, la variable est ajoutée à la fin de blob et la taille du BLOB est redimensionnée en conséquence. A l'aide du paramètre optionnel *, vous pouvez stocker les unes derrière les autres autant de variables ou de listes (cf. les autres commandes BLOB) que vous voulez dans un BLOB, la seule limite étant celle de la mémoire disponible.

Si vous ne passez pas le paramètre optionnel * ni de variable dans le paramètre offset, variable est stockée à partir du début du BLOB en écrasant son contenu précédent. La taille du BLOB est redimensionnée en conséquence.

Si vous passez la variable offset en paramètre, la variable est écrite dans le BLOB à l'offset (à partir de zéro) spécifié par offset. Quel que soit l'endroit où vous placez la variable, la taille du BLOB sera augmentée si nécessaire en fonction de l'emplacement que vous avez défini (ainsi que de la taille de la variable). Les octets redéfinis (autres que ceux que vous venez d'écrire) sont initialisés à la valeur zéro.

Après l'exécution de la commande, la variable du paramètre offset est incrémentée du nombre d'octets ayant été écrits. Vous pouvez par conséquent réutiliser la même variable avec une autre commande d'écriture de BLOB afin de placer une autre variable ou liste juste après celle que vous venez d'écrire.

VARIABLE VERS BLOB accepte tous les types de variables (y compris d'autres BLOBs), à l'exception des types suivants :
- Pointeurs
- Tableaux de pointeurs
- Tableaux à deux dimensions

Notez cependant que si vous stockez une variable de type Entier long qui est une référence à une liste hiérarchique (ListRef), VARIABLE VERS BLOB stockera la variable Entier long, pas la liste. Pour stocker et récupérer des listes hiérarchiques dans un BLOB, utilisez les commandes LISTE VERS BLOB et BLOB vers liste.

ATTENTION : Si vous utilisez un BLOB pour stocker les variables, utilisez par la suite la commande BLOB VERS VARIABLE pour récupérer le contenu du BLOB car les variables sont stockées dans les BLOBs avec un format interne à 4D.

La variable OK prend la valeur 1 si la variable a été correctement stockée. Si l'opération n'a pas pu être effectuée à cause d'un manque de mémoire, la variable OK prend la valeur 0.

Note sur l'indépendance de plate-forme : VARIABLE VERS BLOB et BLOB VERS VARIABLE utilisent un format interne à 4D pour gérer les variables stockées dans les BLOBs. Vous n'avez donc pas besoin de vous préoccuper de la conversion des octets ("byte swapping") entre les différentes plates-formes lors de l'utilisation de ces deux commandes. Un BLOB créé sous Windows à l'aide de ces deux commandes peut être réutilisé sans la moindre manipulation sous Mac OS et vice-versa.

Exemples

(1) Les méthodes projet suivantes vous permettent de stocker et de récupérer rapidement des variables dans les documents sur disque :

```
    ` Méthode projet STOCKER VARIABLES
    ` STOCKER VARIABLES ( Alpha ; Pointeur )
    ` STOCKER VARIABLES ( Document ; -> Tableau )
C_ALPHA (255;$1)
C_POINTEUR ($2)
C_BLOB ($vxDonnéesTableau)
VARIABLE VERS BLOB ($2->;$vxDonnéesTableau)  ` Stocker le tableau dans le BLOB
COMPRESSER BLOB ($vxDonnéesTableau)  ` Compresser le BLOB
BLOB VERS DOCUMENT ($1;$vxDonnéesTableau)  ` Enregistrer le BLOB sur disque

    ` Méthode projet CHARGER VARIABLES
    ` CHARGER VARIABLES ( Alpha ; Pointeur )
    ` CHARGER VARIABLES ( Document ; -> Tableau )
C_ALPHA (255;$1)
C_POINTEUR ($2)
C_BLOB ($vxDonnéesTableau)
DOCUMENT VERS BLOB ($1;$vxDonnéesTableau)  ` Charger le BLOB du disque
DECOMPRESSER BLOB ($vxDonnéesTableau)  ` Décompresser le BLOB
BLOB VERS VARIABLE ($vxDonnéesTableau;$2->)  ` Récupérer le tableau du BLOB
```

Lorsque ces méthodes ont été ajoutées à votre application, vous pouvez écrire :

```
TABLEAU ALPHA (...;taToutTableau;...)
    ` ...
STOCKER VARIABLES ( $vaNomDoc;->taToutTableau)
    ` ...
CHARGER VARIABLES ( $vaNomDoc;->taToutTableau)
```

(2) Les deux méthodes projet suivantes vous permettent de stocker et de récupérer des variables dans un BLOB :

```
    ` Méthode projet STOCKER VARIABLES DANS BLOB
    ` STOCKER VARIABLES DANS BLOB ( Pointeur { ; Pointeur ... { ;  Pointeur } } )
    ` STOCKER VARIABLES DANS BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
C_POINTEUR ($1)
C_ENTIER LONG ($vlParam)

FIXER TAILLE BLOB ($1->;0)
Boucle ($vlParam;2;Nombre de parametres)
    VARIABLE VERS BLOB (${$vlParam}->;$1->;*)
Fin de boucle
```

```
    ` Méthode projet RECUPERER VARIABLES DANS BLOB
    ` RECUPERER VARIABLES DANS BLOB ( Pointeur { ; Pointeur ... { ;  Pointeur } } )
    ` RECUPERER VARIABLES DANS BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
C_POINTEUR ($1)
C_ENTIER LONG ($vlParam;$vlOffset)

$vlOffset:=0
Boucle ($vlParam;2;Nombre de parametres)
    BLOB VERS VARIABLE ($1->;${$vlParam}->;$vlOffset)
Fin de boucle
```

Lorsque ces méthodes ont été ajoutées à votre application, vous pouvez écrire :

```
STOCKER VARIABLES DANS BLOB ( ->vxBLOB;->vgImage;->taTableau1;->taTableau2)
    ` ...
RECUPERER VARIABLES DANS BLOB ( ->vxBLOB;->vgImage;->taTableau1;->taTableau2)
```

Référence

BLOB vers liste, BLOB VERS VARIABLE, LISTE VERS BLOB.

Variables et ensembles système

La variable OK prend la valeur 1 si la variable a été correctement stockée, sinon elle prend la valeur 0.

4

Booléens

Les fonctions booléennes de 4D sont utilisées pour les opérations de type booléennes, c'est-à-dire ne traitant que deux valeurs, VRAI ou FAUX. Ces fonctions sont les suivantes :

- Vrai
- Faux
- Non

L'exemple suivant retourne Vrai dans la variable monBooléen si l'utilisateur a cliqué sur le bouton monBouton et Faux s'il n'a pas cliqué dessus. Lorsqu'un bouton reçoit un clic, la variable du bouton prend la valeur 1. Dans cet exemple, la valeur de la variable booléenne est basée sur la valeur d'un bouton.

```
Si (monBouton=1)  ` Si le bouton a reçu un clic
   monBooléen:=Vrai  ` monBooléen prend la valeur Vrai
Sinon  ` Si le bouton n'a pas reçu de clic,
   monBooléen:=Faux  ` monBooléen prend la valeur Faux
Fin de si
```

L'exemple ci-dessus peut être simplifié et écrit en une seule ligne, plus efficace :

```
monBooléen:=(monBouton=1)
```

Référence

Faux, Non, Opérateurs logiques, Vrai.

Faux → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Faux

Description

Faux retourne la valeur booléenne Faux.

Exemple

L'exemple suivant met la variable vbOptions à Faux :

```
vbOptions:=Faux
```

Référence

Non, Vrai.

Non (booléen) → Booléen

Paramètre	Type		Description
booléen	Booléen	→	Valeur booléenne à inverser
Résultat	Booléen	←	Inverse de booléen

Description

La fonction Non retourne la valeur inverse de booléen, changeant un Vrai en Faux ou un Faux en Vrai.

Exemples

Dans l'exemple suivant, la valeur Vrai est assignée à une variable. Cette valeur est alors modifiée en Faux puis de nouveau en Vrai :

```
Résultat:= Vrai ` Résultat prend la valeur VRAI
Résultat:= Non (Résultat) ` Résultat prend la valeur FAUX
Résultat:= Non (Résultat) ` Résultat prend la valeur VRAI
```

Référence

Faux, Vrai.

Vrai → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai

Description

Vrai retourne la valeur booléenne Vrai.

Exemple

L'exemple suivant met la variable vbOptions à Vrai :

```
vbOptions:=Vrai
```

Référence

Faux, Non.

5

Chaînes de caractères

Introduction

Les symboles d'indice de chaîne sont les suivants :

```
[[...]] sous Windows
≤...≥ ou [[...]] sous MacOS
```

Ces symboles sont utilisés pour désigner un caractère particulier dans une chaîne. Cette syntaxe vous permet de référencer un caractère dans un champ ou une variable de type Alpha ou Texte.

Note : Sous Mac OS, vous obtenez les deux premiers symboles en appuyant sur les touches **Option**+< et **Option**+>.

Lorsque les symboles d'indice de chaîne sont placés à gauche de l'opérateur d'affectation (:=), un caractère est affecté à la position référencée dans la chaîne. Par exemple, en postulant que la chaîne vsNom n'est pas une chaîne vide, le code suivant passe le premier caractère de la chaîne vsNom en majuscule :

```
Si (vsNom#"")
    vsNom[[1]]:=Majusc(vsNom[[1]])
Fin de si
```

Lorsque les symboles d'indice de chaîne apparaissent dans une expression, ils retournent le caractère auquel ils font référence sous la forme d'une chaîne d'un caractère. En voici un exemple :

```
    ` L'exemple suivant teste si le dernier caractère de vtText est le caractère "@"
Si (vtText # "")
    Si (Code de caractere(Sous chaine(vtText;Longueur(vtText);1))=Arobase)
        ` ...
    Fin de si
Fin de si
```

```
    ` En utilisant la syntaxe des caractères d'indice de chaîne, vous écririez plus simplement :
Si (vtText # "")
    Si (Code de caractere(vtText[[Longueur(vtText)]]=Arobase)
        ` ...
    Fin de si
Fin de si
```

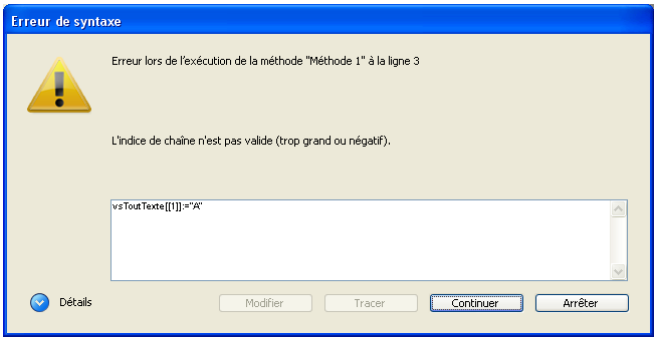
Note avancée sur la référence à des caractères invalides

Lorsque vous utilisez les symboles d'indice de chaîne, **il est de votre responsabilité** de vous référer à des caractères existant dans la chaîne, de la même manière que pour les éléments d'un tableau. Si, par exemple, vous référencez le 20e caractère d'une chaîne, cette chaîne doit contenir au moins 20 caractères.

- Ne pas respecter cette condition en mode interprété n'est pas signalé comme une erreur par 4D.
- Ne pas respecter cette condition en mode compilé (sans options) peut entraîner une "corruption" de la mémoire, si, par exemple, vous écrivez un caractère au-delà de la fin d'une chaîne ou d'un texte.
- Ne pas respecter cette condition en mode compilé est signalé lorsque le contrôle d'exécution est activé. Si, par exemple, vous exécutez le code suivant :

```
    ` Ne pas faire ça !
vsToutTexte:=""
vsToutTexte[[1]]:="A"
```

L'alerte suivante s'affichera :



Exemples

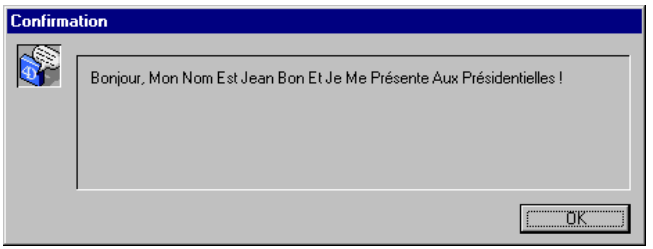
La méthode projet suivante ajoute une lettre capitale à tous les mots du texte passé en paramètre et retourne le texte modifié :

```
      ` Méthode projet de passage en capitale
      ` PasserEnCap ( Texte ) -> Texte
      ` PasserEnCap ( Texte source ) -> Texte avec des lettres capitales

$0:=$1
$vlLen:=Longueur($0)
Si ($vlLen>0)
   $0[[1]]:=Majusc($0[[1]])
   Boucle ($vlChar;1;$vlLen-1)
      Si (Position($0[[$vlChar]];" !&()-{}:;<>?/,.=+*")>0)
         $0[[$vlChar+1]]:=Majusc($0[[$vlChar+1]])
      Fin de si
   Fin de boucle
Fin de si
```

Une cette méthode placée dans la base, la ligne :

```
ALERTE(PasserEnCap ("Bonjour, mon nom est Jean Bon et je me présente aux
                                                          présidentielles !"))
```

... affiche l'alerte suivante :



Référence

Caractere, Code de caractere, Codes ASCII.

Caractere (codeCaractère) → Chaîne

Paramètre	Type		Description
codeCaractère	Numérique	→	Code de caractère
Résultat	Chaîne	←	Caractère représenté par codeCaractère

Description

La fonction Caractere retourne le caractère dont le code est codeCaractère.

• Si la base fonctionne en mode Unicode (mode par défaut pour les bases créées à compter de la version 11 de 4D), vous devez passer une valeur UTF-16 (comprise entre 1 et 65535) dans codeCaractère.
• Si la base fonctionne en mode compatibilité ASCII, vous devez passer un code ASCII (compris entre 0 et 255) dans codeCaractère.
Pour plus d'informations sur les modes de gestion des chaînes de 4D, reportez-vous à la section Codes ASCII.

Important : En mode compatibilité ASCII, toutes les valeurs de texte, champs ou variables, utilisent la table ASCII de Mac OS, sur les plates-formes Macintosh et Windows — si aucune conversion vers une autre table ASCII n'a été effectuée. Pour plus d'informations sur ce point, reportez-vous à la section Codes ASCII.

Astuce : La fonction Caractere est généralement utilisée pour insérer dans l'éditeur de méthodes des caractères qui ne peuvent être saisis au clavier ou des caractères de contrôle.

Exemple

L'exemple suivant utilise la fonction Caractere pour insérer un retour chariot dans une boîte de dialogue d'alerte afin de séparer deux lignes d'information :

```
ALERTE ("Employés : "+Chaine(Enregistrements dans table([Employés]))+Caractere
(Retour chariot)+"Cliquez sur OK pour continuer.")
```

Référence

Code de caractere, Codes ASCII, Codes Unicode, Symboles d'indice de chaîne.

Chaine (expression{; format}) → Chaîne

Paramètre	Type		Description
expression		→	Expression à convertir en chaîne (peut être de type Réel, Entier, Entier long, Date, Heure, Alpha, Texte ou Booléen)
format	Alpha | Num	→	Format d'affichage
Résultat	Chaîne	←	expression convertie en chaîne alphanumérique

Description

La commande Chaine retourne sous forme de chaîne alphanumérique l'expression de type numérique, Date, Heure, chaîne ou Booléen que vous avez passée dans le paramètre expression.

Si vous ne passez pas le paramètre optionnel format, la chaîne est retournée dans le format par défaut du type de données correspondant. Si vous passez le paramètre format, vous pouvez définir suivant vos besoins le formatage de la chaîne retournée.

Expressions numériques

Si expression est du type numérique (Réel, Entier, Entier long), vous pouvez passer le paramètre optionnel de formatage de la chaîne. Voici quelques exemples :

Exemple	Résultat
Chaine(2^15) `Utiliser format défaut	32768 (Format par défaut)
Chaine(2^15;"### ##0 habitants")	32 768 habitants
Chaine(1/3;"##0,00000")	0,33333
Chaine(1/3) `Utiliser format défaut	0,3333333333333330000 (Format défaut)
Chaine(Arctan(1)*4)	3,14159265358979000 (Format défaut)
Chaine(Arctan(1)*4;"##0,00")	3,14
Chaine(-1;"&x")	0xFFFFFFFF
Chaine(-1;"&$")	$FFFFFFFF
Chaine(0 ?+ 7;"&x")	0x0080
Chaine(0 ?+ 7;"&$")	$80
Chaine(0 ?+ 14;"&x")	0x4000
Chaine(0 ?+ 14;"&$") $4000	

Chaine(Num(1=1);"Vrai;;Faux")	Vrai
Chaine(Num(1=2);"Vrai;;Faux")	Faux

Le format est défini de la même manière que pour un champ numérique dans un formulaire. Pour plus d'informations sur le formatage des numériques, reportez-vous au manuel *Mode Développement* de 4D. Vous pouvez également passer le nom d'un style personnalisé dans format. Dans ce cas, le nom du style doit être précédé du caractère “|”.

Expressions de type Date

Si expression est de type Date, la chaîne est retournée dans le format par défaut défini dans le système. Vous pouvez passer dans le paramètre format une des constantes suivantes (thème "Formats d'affichage des dates") :

Numéro	Constante	Exemple
1	Système date court	06/12/2006
2	Système date abrégé	mer. 25 déc. 2006
3	Système date long	mercredi 6 décembre 2006
4	Interne date court spécial	06/12/06 (mais 06-12-1896 ou 06-12-2096)
5	Interne date long	6 décembre 2006
6	Interne date abrégé	6 déc 1996
7	Interne date court	06/12/2006
8	ISO Date	2006-06-12T00:00:00
100	Vide si date nulle	"" au lieu de 0

Notes :

• Le format ISO Date correspond à la norme ISO8601. Ce format contient en principe une date et une heure. Par exemple, la date du 31 mai 2006 à 13h20 est notée 2006-05-31T13:20:00. Il est utilisé lors des traitements XML et dans le cadre des Web services. 4D ne permet pas de stocker dans un seul champ une date et une heure. Toutefois, il est possible de gérer des dates dans ce format à l’aide de la commande Chaine.

• La constante Vide si date nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit retourner une chaîne vide au lieu de zéros.

Voici quelques exemples (en supposant que nous soyons jeudi 5 mars 2009) :

```
$vsRésultat:=Chaine(Date du jour)  ` $vsRésultat prend la valeur "05/03/09"
$vsRésultat:=Chaine(Date du jour;Interne date long)
    ` $vsRésultat prend la valeur "5 Mars 2009"
$vsRésultat:=Chaine(Date du jour;ISO Date)
    ` $vsRésultat prend la valeur "2009-03-05T00:00:00"
```

Expressions de type Heure
Si expression est de type Heure, la chaîne est retournée dans le format par défaut hh:mm:ss. Vous pouvez passer dans le paramètre format une des constantes suivantes (thème "Formats d'affichage des heures") :

Numéro	Constante	Exemple
1	h mn s	01:02:03
2	h mn	01:02
3	Heures Minutes Secondes	1 heure 2 minutes 3 secondes
4	Heures Minutes	1 heure 2 minutes
5	h mn Matin Après midi	1:02 du matin
6	mn s	62:03
7	Minutes secondes	62 minutes 3 secondes
8	ISO Heure	0000-00-00T01:02:03
9	Système heure court	01:02:03
10	Système heure long abrégé	1•02•03 AM (*Mac uniquement*)
11	Système heure long	1:02:03 AM HNEC (*Mac uniquement*)
100	Vide si heure nulle	"" au lieu de 0

Notes :
• Pour plus d'informations sur le format ISO Heure, reportez-vous à la note ci-dessus.
• La constante Vide si heure nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit retourner une chaîne vide au lieu de zéros.

Voici quelques exemples (en supposant qu'il soit 17h30 et 45 secondes) :

```
$vsRésultat:=Chaine(Heure courante)  ` $vsRésultat prend la valeur "17:30:45"
$vsRésultat:=Chaine(Heure courante;Heures Minutes Secondes)
    ` $vsRésultat prend la valeur "17 heures 30 minutes 45 secondes"
```

Expressions de type chaîne
Si expression est de type Alpha ou Texte, la commande retourne la même valeur que celle passée en paramètre. Ce fonctionnement est utile notamment dans le cadre d'une programmation générique utilisant des pointeurs.
Dans ce cas, le paramètre format, s'il est passé, est ignoré.

Expressions de type Booléen
Si expression est de type Booléen, la commande retourne la chaîne "Vrai" ou "Faux" dans la langue de l'application ("True" ou "False" dans une version anglaise de 4D).
Dans ce cas, le paramètre format, s'il est passé, est ignoré.

Référence

Chaine heure, Date, Num.

Code de caractere (unCaractère) → Numérique

Paramètre	Type		Description
unCaractère	Alpha	→	Caractère dont vous voulez obtenir le code
Résultat	Numérique	←	Code du caractère

Description

La commande Code de caractere retourne le code de caractère courant de unCaractère.
• Si la base fonctionne en mode Unicode (mode par défaut pour les bases créées à compter de la version 11 de 4D), la commande retourne le code Unicode UTF-16 de caractère (compris entre 1 et 65535).
• Si la base fonctionne en mode compatibilité ASCII, la commande retourne le code ASCII de unCaractère (compris entre 0 et 255).
Pour plus d'informations sur les modes de gestion des chaînes de 4D, reportez-vous à la section Codes ASCII.

Si la chaîne unCaractère comporte plus d'un caractère, Code de caractere retourne uniquement le code du premier caractère.

La fonction Caractere est l'inverse de Code de caractere. Elle retourne le caractère désigné par un code UTF-16 ou ASCII.

Important : En mode compatibilité ASCII, toutes les valeurs de texte, champs ou variables, utilisent la table ASCII de Mac OS, sur les plates-formes Macintosh et Windows — si aucune conversion vers une autre table ASCII n'a été effectuée. Pour plus d'informations sur ce point, reportez-vous à la section Codes ASCII.

Exemples

(1) Les caractères majuscules et minuscules ne sont pas différenciés lors d'une comparaison ou d'une recherche. Vous pouvez utiliser la fonction Code de caractere si vous souhaitez établir une distinction entre les caractères majuscules et les minuscules.
En effet, cette ligne retourne VRAI :

```
("A" = "a")
```

En revanche, cette ligne retourne FAUX :

```
(Code de caractere("A") = Code de caractere ("a"))
```

(2) L'exemple suivant retourne le code du premier caractère de la chaîne "ABC" :

```
RécupCode := Code de caractere ("ABC")
   ` RécupCode prend la valeur 65, le code de caractère de A
```

(3) Le code suivant :

```
Boucle($vlCar;1;Longueur(vtText))
   Au cas ou
      : (vtText[[$vlCar]]=Caractere(Retour chariot))
         ` Faire quelque chose
      : (vtText[[$vlCar]]=Caractere(Tab))
         ` Faire autre chose
      : (...)
         ` ...
   Fin de cas
Fin de boucle
```

... lorsqu'il est utilisé de nombreuses fois avec des textes de taille importante, s'exécutera plus vite, une fois compilé, s'il est écrit ainsi :

```
Boucle($vlCar;1;Longueur(vtText))
   $vlCode:=Code de caractere(vtText[[$vlCar]])
   Au cas ou
      : ($vlCode=Retour chariot)
         ` Faire quelque chose
      : ($vlCode=Tabulation)
         ` Faire autre chose
      : (...)
         ` ...
   Fin de cas
Fin de boucle
```

... et ce, pour deux raisons principales : il ne référence un caractère qu'une seule fois par itération, et compare des entiers longs et non des chaînes de caractères lorsqu'il teste la présence de retours chariot et de tabulations. Nous vous conseillons d'employer cette technique lorsque vous travaillez avec des caractères standard tels que des Retours chariot et des Tabulations.

Référence

Caractere, Codes ASCII, Symboles d'indice de chaîne.

Note préliminaire

Cette commande répond à des besoins spécifiques liés aux jeux de caractères non romans. Son usage est très restreint.

Convertir caracteres (chaîne; target; srcMask) → Chaîne

Paramètre	Type		Description
chaîne	Chaîne	→	Chaîne à convertir
target	Entier	→	Type de conversion
srcMask	Entier long	→	Partie à convertir
Résultat	Chaîne	←	Chaîne convertie

Description

La commande Convertir caracteres fait directement appel à la fonction TransliterateText du Script Manager d'Apple, permettant de convertir du texte en différentes sous-variantes sur des systèmes non romans. Pour une description complète de cette fonction, reportez-vous à l'adresse :
http://developer.apple.com/documentation/mac/Text/Text-402.html

Les paramètres de cette commande correspondent à ceux de la fonction TransliterateText. A noter que 4D utilise le scriptCode smSystemScript.

CONVERTIR DEPUIS TEXTE (texte4D; jeuCaractères; blobConverti)

Paramètre	Type		Description
texte4D	Chaîne	→	Texte exprimé dans le jeu de caractères courant de 4D
jeuCaractères	Chaîne | Entier long	→	Nom ou Numéro de jeu de caractères
blobConverti	BLOB	←	BLOB contenant le texte converti

Description

La commande CONVERTIR DEPUIS TEXTE permet de convertir un texte exprimé dans le jeu de caractères courant de 4D en un texte exprimé dans un autre jeu de caractères.

Passez dans le paramètre texte4D le texte devant être converti. Ce texte est exprimé dans le jeu de caractères de 4D. Dans les bases de données créée à partir de la version 11, 4D utilise le jeu de caractères Unicode par défaut.

Passez dans jeuCaractères le jeu de caractères à utiliser pour la conversion. Vous pouvez passer une chaîne contenant le nom standard du jeu (par exemple “ISO-8859-1” ou “UTF-8”) ou son identifiant *MIBEnum*. Voici la liste des jeux de caractères pris en charge par les commandes CONVERTIR DEPUIS TEXTE et Convertir vers texte :

MIBEnum	Nom(s)
1017	UTF-32
1018	UTF-32BE
1019	UTF-32LE
1015	UTF-16
1013	UTF-16BE
1014	UTF-16LE
106	UTF-8
1012	UTF-7
3	US-ASCII
3	ANSI_X3.4-1968
3	ANSI_X3.4-1986
3	ASCII
3	cp367
3	csASCII
3	IBM367
3	iso-ir-6
3	ISO_646.irv:1991
3	ISO646-US
3	us
2011	IBM437

2011	cp437
2011	437
2011	csPC8CodePage437
2028	ebcdic-cp-us
2028	cp037
2028	csIBM037
2028	ebcdic-cp-ca
2028	ebcdic-cp-n
2028	ebcdic-cp-wt
2028	IBM037
2027	MacRoman
2027	x-mac-roman
2027	mac
2027	macintosh
2027	csMacintosh
2252	windows-1252
1250	MacCE
1250	x-mac-ce
2250	windows-1250
1251	x-mac-cyrillic
2251	windows-1251
1253	x-mac-greek
2253	windows-1253
1254	x-mac-turkish
2254	windows-1254
1256	x-mac-arabic
2256	windows-1256
1255	x-mac-hebrew
2255	windows-1255
1257	x-mac-ce
2257	windows-1257
17	Shift_JIS
17	csShiftJIS
17	MS_Kanji
17	Shift-JIS
39	ISO-2022-JP
39	csISO2022JP
2026	Big5
2026	csBig5
38	EUC-KR
38	csEUCKR
2084	KOI8-R
2084	csKOI8R
4	ISO-8859-1
4	CP819

4	csISOLatin1
4	IBM819
4	iso-ir-100
4	ISO_8859-1
4	ISO_8859-1:1987
4	l1
4	latin1
5	ISO-8859-2
5	csISOLatin2
5	iso-ir-101
5	ISO_8859-2
5	ISO_8859-2:1987
5	l2
5	latin2
6	ISO-8859-3
6	csISOLatin3
6	ISO-8859-3:1988
6	iso-ir-109
6	ISO_8859-3
6	l3
6	latin3
7	ISO-8859-4
7	csISOLatin4
7	ISO-8859-4:1988
7	iso-ir-110
7	ISO_8859-4
7	l4
7	latin4
8	ISO-8859-5
8	csISOLatinCyrillic
8	cyrillic
8	ISO-8859-5:1988
8	iso-ir-144
8	ISO_8859-5
9	ISO-8859-6
9	arabic
9	ASMO-708
9	csISOLatinArabic
9	ECMA-114
9	ISO-8859-6:1987
9	iso-ir-127
9	ISO_8859-6
10	ISO-8859-7
10	csISOLatinGreek
10	ECMA-118

10	ELOT_928
10	greek
10	greek8
10	iso-ir-126
10	ISO_8859-7
10	ISO_8859-7:1987
11	ISO-8859-8
11	csISOLatinHebrew
11	hebrew
11	iso-ir-138
11	ISO_8859-8
11	ISO_8859-8:1988
12	ISO-8859-9
12	csISOLatin5
12	iso-ir-148
12	ISO_8859-9
12	ISO_8859-9:1989
12	l5
12	latin5
13	ISO-8859-10
13	csISOLatin6
13	iso-ir-157
13	ISO_8859-10
13	ISO_8859-10:1992
13	l6
13	latin6
109	ISO-8859-13
111	ISO-8859-15
111	Latin-9
2025	GB2312
2025	csGB2312
2025	x-mac-chinesesimp
2025	x-mac-chinesesimp
57	GB_2312-80
57	csISO58GB231280

Note : Plusieurs lignes ont le même identifiant MIBEnum car un jeu de caractères peut avoir plusieurs noms (alias).

Pour plus d'informations sur les noms des jeux de caractères, reportez-vous à l'adresse http://www.iana.org/assignments/character-sets

Après l'exécution de la commande, le texte converti est retourné dans le BLOB blobConverti. Ce BLOB pourra être relu par la commande Convertir vers texte.

Note : Cette commande fonctionne uniquement lorsque 4D est exécuté en mode Unicode (l'option Unicode doit être cochée dans les préférences de 4D, cf. section Codes ASCII). Si elle est utilisée dans le mode de compatibilité (non Unicode), blobConverti est retourné vide et la variable OK prend la valeur 0.

Référence

Convertir vers texte.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable OK prend la valeur 1. Sinon, elle prend la valeur 0.

Convertir vers texte (blob; jeuCaractères) → Texte

Paramètre	Type		Description
blob	BLOB	→	BLOB contenant un texte exprimé dans un jeu de caractères spécifique
jeuCaractères	Chaîne | Entier long	→	Nom ou Numéro du jeu de caractères de blob
Résultat	Texte	←	Contenu de blob exprimé dans le jeu de caractères 4D

Description

La commande Convertir vers texte convertit le texte contenu dans le paramètre blob et le retourne en texte exprimé dans le jeu de caractères de 4D. Dans les bases de données créées à partir de la version 11, 4D utilise le jeu de caractères Unicode par défaut.

Passez dans jeuCaractères le jeu de caractères dans lequel est exprimé le texte contenu dans blob, et qui doit être utilisé pour la conversion. Vous pouvez passer une chaîne fournissant le nom standard du jeu ou l'un de ses alias (par exemple “ISO-8859-1” ou “UTF-8”), ou encore son identifiant (entier long). Pour plus d’informations, reportez-vous à la description de la commande CONVERTIR DEPUIS TEXTE.

Note : Cette commande fonctionne uniquement lorsque 4D est exécuté en mode Unicode (l’option Unicode doit être cochée dans les préférences de 4D, cf. section Codes ASCII). Si elle est utilisée dans le mode de compatibilité (non Unicode), elle retourne une chaîne vide et la variable OK prend la valeur 0.

Référence

CONVERTIR DEPUIS TEXTE.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable OK prend la valeur 1. Sinon, elle prend la valeur 0.

Inserer chaine (source; insertion; position) → Alpha

Paramètre	Type		Description
source	Alpha	→	Chaîne dans laquelle effectuer l'insertion
insertion	Alpha	→	Chaîne à insérer dans source
position	Numérique	→	Position de l'insertion
Résultat	Alpha	←	Chaîne résultante

Description

Inserer chaine insère la chaîne de caractères alphanumériques insertion dans la chaîne source à partir de position et retourne la chaîne de caractères résultante. La chaîne insertion est placée avant le caractère désigné par position.

Si insertion est une chaîne vide (""), Inserer chaine retourne source inchangé.

Si position est supérieur à la longueur de source, insertion est ajouté à la fin de source. Si position est inférieur à un (1), insertion est inséré au début de source.

Inserer chaine est différent de Remplacer caracteres puisque cette fonction insère des caractères au lieu de les remplacer.

Exemples

L'exemple suivant illustre l'utilisation de Inserer chaine. Les résultats sont affectés à la variable vRésultat.

```
vRésultat := Inserer chaine ("L'arbre"; " vert"; 8)  ` vRésultat est égal à "L'arbre vert"
vRésultat := Inserer chaine ("Tale"; "b"; 3)  ` vRésultat est égal à "Table"
vRésultat := Inserer chaine ("Indention"; "ta"; 6)  ` vRésultat est égal à "Indentation"
```

Référence

Remplacer caracteres, Remplacer chaine, Supprimer chaine.

ISO vers Mac (texte) → Chaîne

Paramètre	Type		Description
texte	Chaîne	→	Texte en jeu standard Web
Résultat	Chaîne	←	Texte en ASCII MacOS

Note de compatibilité : Cette commande fonctionne uniquement lorsque la base est exécutée en mode compatibilité ASCII. En mode Unicode, elle ne fait rien (la chaîne texte est retournée sans modification). A compter de la version 11 de 4D, cette commande est donc obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes CONVERTIR DEPUIS TEXTE ou Convertir vers texte.

Description

ISO vers Mac retourne un texte, exprimé à l'aide de la table ASCII Mac OS, équivalent au texte passé dans texte, exprimé à l'aide de la table ISO Latin-1.

Cette commande attend un paramètre de type Texte exprimé à l'aide de la table ISO Latin-1.

Vous n'aurez généralement pas besoin d'utiliser cette commande. 4D convertit, dans les deux sens, les caractères reçus et envoyés par les navigateurs Web. Comme résultat, les valeurs textes que vous gérez, dans un process de connexion Web, sont exprimées à l'aide du filtre Mac OS ASCII.

Dans 4D en mode compatibilité ASCII (Non Unicode), tous les valeurs, champs ou variables de texte que vous n'avez pas convertis à l'aide d'un autre filtre ASCII sont encodés Mac OS sous Macintosh et Windows. Pour de plus amples informations sur ce sujet, référez-vous à la section Codes ASCII.

Sous Windows, vous n'avez pas besoin de vous soucier de la conversion des codes ASCII. Lorsque vous copiez ou collez du texte entre 4D et Windows ou lorsque vous importez/exportez des données, 4D effectue automatiquement ces conversions. Cependant, lorsque vous utilisez des commandes lecture/écriture comme ENVOYER PAQUET ou RECEVOIR PAQUET, il faut faire les conversions ASCII.

Par conséquent, quelle que soit la plate-forme sur laquelle vous travaillez, si vous voulez lire des documents HTML ISO Latin-1 stockés sur disque à l'aide de RECEVOIR PAQUET, vous avez juste besoin de convertir le texte à l'aide de ISO vers Mac. C'est en fait le principal rôle de cette fonction.

Sous Windows, dans ce cas, vous ne devez pas filtrer les caractères à l'aide d'un filtre d'import ASCII.

Exemple

La ligne suivante convertit le texte encodé ISO Latin-1 stocké dans vtTexte en un texte encodé Mac OS :

```
RECEVOIR PAQUET ($vhDocRef;vtTexte;16*1024)
    ` Lire du texte d'un document HTML ISO Latin-1
vtTexte:=ISO vers Mac(vtTexte)
```

Référence

Codes ASCII, Mac vers ISO, RECEVOIR PAQUET, UTILISER FILTRE.

Lire traduction chaine (resName) → Chaîne

Paramètre	Type		Description
resName	Chaîne	→	Nom d'attribut resname
Résultat	Chaîne	←	Valeur de la chaîne désignée par resName dans le langage courant

Description

La commande Lire traduction chaine retourne la valeur de la chaîne désignée par l'attribut resName pour la langue courante.

Cette fonctionne uniquement dans le cadre d'une architecture XLIFF. Pour plus d'informations sur ce type d'architecture, reportez-vous à la description de la prise en charge du XLIFF dans le manuel *Mode Développement*.

Note : La commande Lire langue courante base permet de connaître la langue utilisée par l'application.

Passez dans resName le nom de ressource de la chaîne dont vous voulez obtenir la traduction dans la langue cible courante (target).

A noter que le XLIFF est diacritique.

Exemple

Voici un extrait de fichier .xlf :

```
<file source-language="en-US" target-language="fr-FR">
        [...]
        <trans-unit resname="Show on disk">
            <source>Show on disk</source>
            <target>Montrer sur le disque</target>
        </trans-unit>
```

Après exécution de l'instruction suivante :

```
$valeurFR:=Lire traduction chaine("Show on disk")
```

... si la langue courante est le français, *$valeurFR* contient "Montrer sur le disque".

Référence

Lire langue courante base.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable OK prend la valeur 1. Si resName n'est pas trouvé, la commande retourne une chaîne vide et la variable OK prend la valeur 0.

Longueur (chaîne) → Numérique

Paramètre	Type		Description
chaîne	Alpha	→	Chaîne dont vous voulez connaître la longueur
Résultat	Numérique	←	Nombre de caractères de chaîne

Description

Longueur vous permet d'obtenir la longueur de chaîne. Longueur retourne le nombre de caractères alphanumériques contenus dans chaîne.

Note : Le test Si(vTexte="") équivaut au test Si(Longueur(vTexte)=0).

Exemples

L'exemple suivant illustre l'utilisation de Longueur. Les valeurs retournées sont assignées à la variable vRésultat. Les commentaires fournissent la valeur de vRésultat :

```
vRésultat := Longueur ("Topaze") ` vRésultat prend la valeur 6
vRésultat := Longueur ("Citoyen") ` vRésultat prend la valeur 7
```

Mac vers ISO (texte) → Chaîne

Paramètre	Type		Description
texte	Chaîne	→	Texte en ASCII Mac OS
Résultat	Chaîne	←	Texte en jeu standard Web

Note de compatibilité : Cette commande fonctionne uniquement lorsque la base est exécutée en mode compatibilité ASCII. En mode Unicode, elle ne fait rien (la chaîne texte est retournée sans modification). A compter de la version 11 de 4D, cette commande est donc obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes CONVERTIR DEPUIS TEXTE ou Convertir vers texte.

Description

Mac vers ISO retourne un texte équivalent à celui passé dans le paramètre texte mais exprimé à l'aide de la table de caractères Web définie dans le menu **Jeu standard** de la page Web/Options des Préférences de l'application.

Cette commande attend un paramètre de type texte exprimé à l'aide de la table ASCII Mac OS.

Vous n'aurez généralement pas besoin d'utiliser cette commande. 4D convertit, dans les deux sens, les caractères reçus et envoyés par les navigateurs Web. En résultat, les valeurs textes que vous manipulez, à l'intérieur d'un process de connexion Web, sont toutes exprimées à l'aide de la table ASCII Mac OS.

Dans 4D en mode compatibilité ASCII (Non Unicode), chaque valeur, champ ou variable de texte est encodé sur la base de la table ASCII Mac OS sous Macintosh et Windows, dans la mesure où vous ne les avez pas convertis à l'aide d'un filtre ASCII. Pour plus d'informations sur ce point, référez-vous à la section Codes ASCII.

Sous Windows, vous n'avez pas besoin de vous préoccuper de la conversion des codes ASCII. Lorsque vous copiez ou collez du texte entre 4D et Windows ou lorsque vous importez/exportez des données, 4D effectue automatiquement ces conversions. Cependant, lorsque vous utilisez des commandes de lecture/écriture telles que ENVOYER PAQUET ou RECEVOIR PAQUET, 4D n'effectue aucune conversion de code ASCII.

Par conséquent, quelle que soit la plate-forme sur laquelle vous travaillez, si vous voulez écrire sur disque des documents HTML ISO Latin-1 ou utilisant d'autres jeux de caractères Web, vous avez juste besoin de convertir le texte à l'aide de la fonction Mac vers ISO. C'est en fait le principal rôle de cette commande.

Sous Windows, dans ce cas, vous ne devez pas filtrer les caractères à l'aide d'un filtre d'exportation ASCII.

Exemples

(1) La ligne suivante convertit par défaut le texte encodé Mac OS stocké dans vtTexte en texte encodé ISO-Latin 1 :

```
vtTexte:=Mac vers ISO(vtTexte)
```

(2) Lors du développement d'une application 4D Web Server, vous créez par programmation des pages HTML que vous enverrez par la suite sur Intranet ou Internet à l'aide de la commande ENVOYER FICHIER HTML. Dans certains de ces documents se trouvent des références ou des liens vers d'autres documents. La méthode projet ci-dessous calcule le chemin d'accès HTML à partir du chemin d'accès Windows ou Macintosh reçu en paramètre :

```
` Méthode projet Chemin HTML
` Chemin HTML ( Texte ) -> Texte
` Chemin HTML ( Chemin d'accès système natif ) -> Chemin d'accès HTML

C_TEXTE($0;$1)
C_ENTIER LONG($vlCar;$vlAscii)
C_ALPHA(31;$vsCar)

$0:=""
Si (Sous Windows )
   $1:=Remplacer chaine($1;"\";"/")
Sinon
   $1:=Remplacer chaine($1;":";"/")
Fin de si
$1:=Mac vers ISO($1)
Boucle ($vlCar;1;Longueur($1))
   $vlAscii:=Code de caractere($1[[$vlCar]])
   Au cas ou
      : ($vlAscii>=127)
         $vsCar:="%"+Sous chaine(Chaine($vlAscii;"&$");2)
      : (Position(Caractere($vlAscii);":<>&%= "+Caractere(34))>0)
                  $vsCar:="%"+Sous chaine(Chaine($vlAscii;"&$");2)
```

```
        Sinon
            $vsCar:=Caractere($vlAscii)
        Fin de cas
        $0:=$0+$vsCar
    Fin de boucle
```

Note : La méthode projet Sous Windows est listée dans la section Présentation des documents système.

Une fois cette méthode projet placée dans votre base, si vous voulez inclure une liste de liens FTP vers des documents présents dans un répertoire particulier, vous pouvez écrire par exemple :

```
    ` Variables interprocess définies, par exemple, dans la méthode base Sur ouverture
<>vsFTPURL:="ftp://123.4.56.78/Spiders/"
<>vsFTPDirectory:="APS500:Spiders:" ` Ici, un chemin du gestionnaire de fichier Mac OS
    ` ...

    ` ...
TABLEAU ALPHA(31;$asDocuments;0)
LISTE DES DOCUMENTS(...;$asDocuments)
$vlNbDocuments:=Taille tableau($asDocuments)
jsHandler:=...
Boucle ($vlDocument;1;$vlNbDocuments)
    vtHTMLCode:=vtHTMLCode+"<P><A HREF="+Caractere(34)+<>vsFTPURL+
                                Chemin HTML (Sous chaine ($1+$asDocuments{$vlDocument};
                            Longueur (<>vsFTPDirectory)+1)) + Caractere(34)+ jsHandler+">
                                "+$asDocuments{$vlDocument}+"</A></P>"+Caractere(13)
Fin de boucle
    ` ...
```

Référence

Codes ASCII, ENVOYER FICHIER HTML, ENVOYER PAQUET, ISO vers Mac, UTILISER FILTRE.

Mac vers Windows (texte) → Chaîne

Paramètre	Type		Description
texte	Chaîne	→	Texte exprimé en ASCII Mac OS
Résultat	Chaîne	←	Texte exprimé en ANSI Windows

Note de compatibilité : Cette commande fonctionne uniquement lorsque la base est exécutée en mode compatibilité ASCII. En mode Unicode, elle ne fait rien (la chaîne texte est retournée sans modification). A compter de la version 11 de 4D, cette commande est donc obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes CONVERTIR DEPUIS TEXTE ou Convertir vers texte.

Description

Mac vers Windows retourne un texte exprimé avec la table ANSI Windows équivalent au texte passé dans texte, exprimé avec la table ASCII Mac OS.

Cette commande attend un paramètre de type Texte exprimé en ASCII Mac OS.

En général, vous n'avez pas besoin d'utiliser cette commande. Dans 4D en mode compatibilité ASCII (Non Unicode), chaque valeur, champ ou variable de texte est encodée sur la base de la table ASCII Mac OS sous Macintosh et Windows, dans la mesure où vous ne l'avez pas converti(e) à l'aide d'un filtre ASCII. Pour plus d'informations sur ce point, référez-vous à la section Codes ASCII.

Sous Windows, vous n'avez pas besoin de vous préoccuper de la conversion des codes ASCII. Lorsque vous copiez ou collez du texte entre 4D et Windows ou lorsque vous importez/exportez des données, 4D effectue automatiquement ces conversions.

Cependant, lorsque vous utilisez des commandes de lecture/écriture telles que ENVOYER PAQUET ou RECEVOIR PAQUET, vous devez explicitement effectuer des conversions ASCII. C'est, en fait, le principal rôle de cette commande. Référez-vous à l'exemple ci-dessous.

Note : La commande remplace les caractères CR (Retour chariot) par des caractères CRLF (Retour chariot + Retour à la ligne, codes de caractères 13 et 10). Par conséquent, le texte retourné peut être plus long que le texte d'origine.

Exemple

Sous Windows, lorsque vous écrivez des caractères dans un document à l'aide de ENVOYER PAQUET, et si vous n'utilisez pas de filtre ASCII d'exportation pour convertir les caractères Mac OS vers Windows (cf. la commande UTILISER FILTRE), il vous faut convertir vous-même le texte de Mac OS vers Windows. Vous pouvez le faire de la manière suivante :

```
` ...
ENVOYER PAQUET ($vhDocRef;Mac vers Windows(vtTexte))
` ...
```

Référence

Codes ASCII, ENVOYER PAQUET, UTILISER FILTRE, Windows vers Mac.

Majusc (laChaîne{; *}) → Alpha

Paramètre	Type		Description
laChaîne	Alpha	→	Chaîne à convertir en majuscules
*	*	→	Si passé : conserver les accents Si omis : supprimer les accents
Résultat	Alpha	←	chaîne en majuscules

Description

Majusc retourne une chaîne de caractères égale à laChaîne dont tous les caractères alphabétiques ont été convertis en majuscules.

Le paramètre facultatif *, s'il est passé, indiqueque les éventuels caractères accentués présents dans laChaîne doivent être retournés sous forme de majuscules accentuées. Par défaut, lorsque ce paramètre est omis, les caractères accentués “perdent” leur accent une fois la conversion effectuée.

Exemples

(1) Cet exemple compare les résultats obtenus suivant que le paramètre * a été passé ou non :

```
$lachaine:=Majusc("hélène")  ` $lachaine vaut « HELENE »
$lachaine:=Majusc("hélène";*)  ` $lachaine vaut « HÉLÈNE »
```

(2) Reportez-vous à l'exemple de Minusc.

Référence

Minusc.

Minusc (laChaîne; *) → Alpha

Paramètre	Type		Description
laChaîne	Alpha	→	Chaîne à convertir en minuscules
*	*	→	Si passé : conserver les accents, Si omis : supprimer les accents
Résultat	Alpha	←	chaîne en minuscules

Description

Minusc retourne une chaîne de caractères égale à laChaîne dont tous les caractères alphabétiques ont été convertis en minuscules.

Le paramètre facultatif *, s'il est passé, indiqueque les éventuels caractères accentués présents dans laChaîne doivent être retournés sous forme de minuscules accentuées. Par défaut, lorsque ce paramètre est omis, les caractères accentués “perdent” leur accent une fois la conversion effectuée.

Exemples

(1) L'exemple suivant est une méthode projet qui met en majuscule (capitale) le premier caractère de la chaîne ou du texte qui lui est passé(e). Par exemple, Nom := Capitale ("jean") donnerait à Nom la valeur "Jean" :

```
` Méthode projet Capitale
` Capitale ( Chaîne ) -> Chaîne
` Capitale ( Tout texte ou chaîne ) -> texte avec une lettre capitale

$0:=Minusc($1)
Si (Longueur($0)>0)
    $0[[1]]:=Majusc($0[[1]])
Fin de si
```

(2) Cet exemple compare les résultats obtenus suivant que le paramètre ***** a été passé ou non :

```
$lachaine:=Minusc("DÉJÀ VU") ` $lachaine vaut « deja vu »
$lachaine:=Minusc("DÉJÀ VU";*) ` $lachaine vaut « déjà vu »
```

Référence

Majusc.

Num (expression{; séparateur}) → Numérique

Paramètre	Type		Description
expression	Chaîne | Booléen | Num	→	Chaîne à convertir en numérique ou Booléen à convertir en 0 ou 1 ou Expression numérique
séparateur	Chaîne	→	Séparateur décimal
Résultat	Numérique	←	Valeur numérique du paramètre expression

Description

La fonction Num retourne sous forme de numérique l'expression de type chaîne, Booléen ou numérique que vous avez passée dans le paramètre expression. Le paramètre facultatif séparateur permet de désigner un séparateur décimal pour l'évaluation des expressions de type chaîne.

Expressions de type chaîne
Si expression ne contient que des caractères alphabétiques, Num retourne zéro. Si expression contient des caractères alphabétiques et des caractères numériques, Num ignore les caractères alphabétiques. Ainsi, Num transformera la chaîne "a1b2c3" en nombre 123.

Note : Seuls les 32 premiers caractères de expression sont pris en compte.

Il existe trois caractères réservés que Num traite de manière particulière. Il s'agit du séparateur décimal tel que défini dans le système (si le paramètre séparateur n'est pas passé), du tiret (-) et du e (ou E). Ils seront interprétés en tant que caractères de formatage des nombres :

• Le séparateur décimal est interprété en tant que tel et doit être inclus dans la chaîne de caractères numériques. Par défaut, la commande utilise le séparateur décimal défini dans le système d'exploitation. Vous pouvez modifier ce caractère à l'aide du paramètre séparateur (cf. ci-dessous).
• Le tiret définit un nombre ou un exposant négatif (signe moins). Le tiret doit être placé devant tout caractère numérique négatif ou derrière le e pour un exposant. Hormis le cas du caractère e, si le tiret est inclus dans une chaîne numérique, la partie de la chaîne se trouvant derrière le tiret est ignorée. Par exemple, Num("123-456") retourne 123, mais Num("-9") retourne -9.

• Le e ou E désigne tout caractère numérique se trouvant à sa droite comme étant la puissance d'un exposant. Le e doit être inclus dans une chaîne numérique. Ainsi, Num ("123e-2") retourne 1,23.
A noter que dans le cas où la chaîne comporte plus d'un caractère e, la conversion pourra donner des résultats différents sous Mac OS et sous Windows.

Le paramètre séparateur permet de désigner un séparateur décimal personnalisé pour l'évaluation de expression. Lorsque la chaîne à évaluer est exprimée avec un séparateur décimal différent du séparateur système, la commande retourne un résultat incorrect. Le paramètre séparateur permet dans ce cas d'obtenir une évaluation correcte. Lorsque ce paramètre est passé, la commande ne tient pas compte du séparateur décimal système. Vous pouvez passer un ou plusieurs caractères.

Note : La commande LIRE FORMATAGE SYSTEME permet de connaître le séparateur décimal courant ainsi que plusieurs autres paramètres système régionaux.

Expressions de type Booléen
Si vous passez une expression booléenne dans le paramètre expression, Num retourne 1 si expression est VRAI, sinon Num retourne 0.

Expressions de type numérique
Si vous passez une expression numérique dans le paramètre expression, Num retourne telle quelle la valeur passée dans le paramètre expression. Ce fonctionnement est utile notamment dans le cadre d'une programmation générique utilisant des pointeurs.

Exemples

(1) L'exemple suivant illustre la manière dont Num fonctionne lorsqu'un argument de type chaîne lui est passé. A chaque ligne, un numérique est assigné à la variable vRésultat. Les commentaires décrivent les résultats :

```
vRésultat := Num ("ABCD") ` vRésultat vaut 0
vRésultat := Num ("A1B2C3") ` vRésultat vaut 123
vRésultat := Num ("123") ` vRésultat vaut 123
vRésultat := Num ("123,4") ` vRésultat vaut 123,4
vRésultat := Num ("-123") ` vRésultat vaut -123
vRésultat := Num ("-123e2") ` vRésultat vaut -12300
```

(2) Dans l'exemple suivant, [Client]Dette est comparé à la valeur 1000. La fonction Num appliquée à cette comparaison retourne 0 ou 1. La multiplication d'une chaîne par 0 ou 1 retourne soit la chaîne, soit une chaîne vide. En définitive, le champ [Client]Risque reçoit la valeur “Acceptable” ou “Inacceptable” :

```
    ` Si le client a des dettes inférieures à 1000, le risque est acceptable.
    ` Si le client a des dettes supérieures à 1000, le risque est inacceptable.
[Client]Risque:=("Acceptable"* Num([Client]Dettes < 1000))+("Inacceptable" *
                                                    Num([Client]Dettes >= 1000))
```

(3) Cet exemple compare les résultats obtenus en fonction du séparateur “courant” :

```
$lachaine:="33,333.33"
$lenum:=Num($lachaine)
    ` par défaut, $lenum vaut 33,33333 sur un système français
$lenum:=Num($lachaine;".")
    ` $lenum vaut bien 33 333,33 quel que soit le système
```

Référence

Chaine, LIRE FORMATAGE SYSTEME, Opérateurs logiques, Opérateurs sur les chaînes.

Position

version 11 (Modifiée)

Position (àChercher; laChaîne{; départ{; longTrouvée{; *}}}) → Numérique

Paramètre	Type		Description
àChercher	Alpha	→	Chaîne à rechercher
laChaîne	Alpha	→	Chaîne dans laquelle effectuer la recherche
départ	Numérique	→	Position dans laChaîne où débuter la recherche
longTrouvée	Entier long	←	Longueur de la chaîne trouvée
*	*	→	Si passé : évaluation basée sur les codes de caractères
Résultat	Numérique	←	Position de la première occurrence de àChercher

Description

Position retourne la position de la première occurrence de àChercher dans laChaîne.

Si laChaîne ne contient pas àChercher, la fonction retourne zéro (0).

Si Position trouve une occurrence de àChercher, la fonction retourne la position du premier caractère de cette occurrence dans laChaîne.
Si vous demandez la position d'une chaîne vide à l'intérieur d'une chaîne vide, Position retourne zéro (0).

Par défaut, la recherche débute au premier caractère de laChaîne. Le paramètre facultatif départ vous permet de préciser le caractère auquel doit démarrer la recherche dans laChaîne.

Le paramètre longTrouvée, s'il est passé, retourne la longueur de la chaîne effectivement trouvée par la recherche. Ce paramètre est nécessaire pour pouvoir gérer correctement les lettres pouvant s'écrire à l'aide d'un ou plusieurs caractères (ex : æ et ae, ß et ss...).
A noter que lorsque le paramètre * est passé (cf. ci-dessous), ces lettres ne sont pas considérées comme équivalentes (æ # ae) ; dans ce mode, longTrouvée est toujours égal à la longueur de àChercher (si une occurrence est trouvée).

Par défaut, la commande effectue des comparaisons globales, tenant compte des particularités linguistiques et des lettres pouvant s'écrire avec un ou plusieurs caractères (par exemple æ = ae). En revanche, elle n'est pas diacritique (a=A, a=à...) et ne tient pas compte des caractères "ignorables" tels que les retours chariot (spécification Unicode).
Pour modifier ce fonctionnement, passez l'astérisque * en dernier paramètre. Dans ce cas, les comparaisons sont effectuées sur la base des codes des caractères. Vous devez donc passer le paramètre * :
• si vous souhaitez tenir compte des caractères spéciaux, utilisés par exemple comme délimiteurs (retours chariot, Caractere(1)...),
• si l'évaluation des caractères doit tenir compte de la casse et des accents (a#A, a#à...).
A noter que dans ce mode, l'évaluation ne gère pas les variations d'écriture des mots.

Attention : Vous ne pouvez pas utiliser le caractère joker (@) avec Position. Si, par exemple, vous passez "abc@" dans àChercher, la fonction recherchera effectivement la chaîne "abc@" et non pas "abc suivi de toute valeur".

Exemples

(1) Les exemples suivants illustrent l'utilisation de Position. Les résultats sont assignés à la variable vRésultat. Les commentaires fournissent la valeur de vRésultat :

```
vRésultat := Position ("ll"; "Billard") ` vRésultat prend la valeur 3
vRésultat := Position (vText1; vText2) ` Position de la première occurrence de vText1 dans
                                                                                    vText2
vRésultat := Position ("day"; "Today is the first day";1) ` vRésultat prend la valeur 3
vRésultat := Position ("day"; "Today is the first day";4) ` vRésultat prend la valeur 20
vRésultat := Position ("DAY"; "Today is the first day";1;*) ` vRésultat prend la valeur 0

vRésultat := Position ("oe"; "Nœud";1;$long) ` vRésultat =2, $long = 1
```

(2) Dans l'exemple suivant, le paramètre longTrouvée permet de rechercher toutes les occurrences de "fluss" dans un texte, quelle que soit l'orthographe du mot :

```
$départ:=1
Repeter
    vRésultat := Position ("fluss";$letexte;$départ;$longtrouvée)
    $départ:= $départ+$longtrouvée
Jusque(vRésultat=0)
```

Référence

Opérateurs de comparaison, Sous chaine.

Remplacer caracteres (source; nouveau; position) → Alpha

Paramètre	Type		Description
source	Alpha	→	Chaîne de départ
nouveau	Alpha	→	Nouveaux caractères
position	Numérique	→	Position de départ du remplacement
Résultat	Alpha	←	Chaîne résultante

Description

Remplacer caracteres retourne une chaîne résultant du remplacement des caractères, dans la chaîne source, à partir de position, par la chaîne nouveau.

Si nouveau est une chaîne vide (""), Remplacer caracteres retourne source inchangé. Remplacer caracteres retourne toujours une chaîne de la même longueur que source. Si position est inférieur ou supérieur à la longueur de source, Remplacer caracteres retourne source.

La fonction Remplacer caracteres est différente de Inserer chaine car elle remplace des caractères au lieu de les insérer.

Exemples

L'exemple suivant illustre l'utilisation de Remplacer caracteres. Les résultats sont affectés à la variable vRésultat.

```
vRésultat := Remplacer caracteres ("Acme"; "CME"; 2)  ` vRésultat est égal à "ACME"
vRésultat := Remplacer caracteres ("novembre";"déc"; 1)  ` vRésultat est égal à "décembre"
```

Référence

Inserer chaine, Remplacer chaine, Supprimer chaine.

Remplacer chaine (source; obsolète; nouveau{; remplacements}{; *}) → Alpha

Paramètre	Type		Description
source	Alpha	→	Chaîne de départ
obsolète	Alpha	→	Caractère(s) à remplacer
nouveau	Alpha	→	Chaîne de remplacement (si chaîne vide, toutes les occurrences sont effacées)
remplacements	Numérique	→	Nombre de remplacements à effectuer
*	*	→	Si passé : évaluation basée sur les codes de caractères
Résultat	Alpha	←	Chaîne résultante

Description

Remplacer chaine retourne une chaîne de caractères résultant du remplacement dans source de obsolète par nouveau.

Si nouveau est une chaîne vide (""), Remplacer chaine supprime chaque occurrence de obsolète dans source.

Si remplacements est spécifié, Remplacer chaine ne remplace que le nombre d'occurrences de obsolète spécifié, à partir du premier caractère de source. Si remplacements est omis, toutes les occurrences de obsolète sont remplacées.

Si obsolète est une chaîne vide, Remplacer chaine retourne source inchangé.

Par défaut, la commande effectue des comparaisons globales, tenant compte des particularités linguistiques et des lettres pouvant s'écrire avec un ou plusieurs caractères (par exemple æ = ae). En revanche, elle n'est pas diacritique (a=A, a=à...) et ne tient pas compte des caractères "ignorables" tels que les retours chariot (spécification Unicode).
Pour modifier ce fonctionnement, passez l'astérisque * en dernier paramètre. Dans ce cas, les comparaisons sont effectuées sur la base des codes des caractères. Vous devez donc passer le paramètre * :
• si vous souhaitez remplacer des caractères spéciaux, utilisés par exemple comme délimiteurs (retours chariot, Caractere(1)...),
• si le remplacement des caractères doit tenir compte de la casse et des accents (a#A, a#à...).
A noter que dans ce mode, l'évaluation ne gère pas les variations d'écriture des mots.

Exemples

(1) L'exemple suivant illustre l'utilisation de Remplacer chaine. Les résultats sont affectés à la variable vRésultat. Les commentaires fournissent la valeur de la variable :

```
vRésultat := Remplacer chaine ("Ville"; "ll"; "d")     ` vRésultat est égal à "Vide"
vRésultat := Remplacer chaine ("Table"; "b"; "")     ` vRésultat est égal à "Tale"
    ` Remplacer toutes les tabulations par des virgules
vRésultat := Remplacer chaine (var; Caractere(Tabulation); ",";*)
```

(2) L'exemple suivant élimine les retours chariot et les tabulations du texte contenu dans la variable vRésultat :

```
vRésultat := Remplacer chaine (Remplacer chaine(vRésultat;Caractere(Retour chariot);
                                                    "";*);Caractere(Tabulation);"";*)
```

(3) L'exemple suivant illustre le rôle du paramètre * dans le cadre d'une évaluation diacritique :

```
vRésultat:=Remplacer chaine("Crème brûlée";"Brulee";"caramel")
    `vRésultat est égal à "Crème caramel"
vRésultat:=Remplacer chaine("Crème brûlée";"Brulee";"caramel";*)
    `vRésultat est égal à "Crème brûlée"
```

Référence

Inserer chaine, Remplacer caracteres, Supprimer chaine.

Sous chaine (source; àPartirDe{; nbCars}) → Alpha

Paramètre	Type		Description
source	Alpha	→	Chaîne de laquelle extraire une sous-chaîne
àPartirDe	Numérique	→	Position du premier caractère
nbCars	Numérique	→	Nombre de caractères à extraire
Résultat	Alpha	←	Sous-chaîne de source

Description

La fonction Sous chaine retourne la partie de source délimitée par les paramètres àPartirDe et nbCars.

Le paramètre àPartirDe indique le premier caractère de la chaîne à retourner, et nbCars définit le nombre de caractères à retourner.

Si nbCars n'est pas défini ou si le total de àPartirDe plus nbCars est supérieur au nombre de caractères de la chaîne source, Sous chaine retourne tous les caractères de la chaîne à partir du caractère spécifié par àPartirDe. Si àPartirDe est supérieur au nombre de caractères de la chaîne, Sous chaine retourne une chaîne vide ("").

Exemples

(1) L'exemple suivant illustre l'utilisation de Sous chaine. Les résultats sont assignés à la variable vRésultat. Les commentaires fournissent la valeur de vRésultat :

```
vRésultat := Sous chaine ("08/04/62"; 4; 2) ` vRésultat prend la valeur "04"
vRésultat := Sous chaine ("Important"; 1; 6) ` vRésultat prend la valeur "Import"
vRésultat := Sous chaine (var; 2) ` vRésultat retourne tous les caractères sauf le premier
```

(2) La méthode projet suivante ajoute au tableau de type texte ou alpha, dont le pointeur est passé en second paramètre, les paragraphes tirés du texte passé en premier paramètre :

```
    ` EXTRAIRE PARAGRAPHES
    ` EXTRAIRE PARAGRAPHES ( Texte ; Pointeur )
    ` EXTRAIRE PARAGRAPHES ( Texte à étudier ; -> Tableau de paragraphes )

C_TEXTE ($1)
C_POINTEUR ($2)

$vlElem:=Taille tableau($2->)
Repeter
    $vlElem:=$vlElem+1
    INSERER DANS TABLEAU($2->;$vlElem)
    $vlPos:=Position(Caractere(Retour chariot);$1)
    Si ($vlPos>0)
        $2->{$vlElem}:=Sous chaine($1;1;$vlPos-1)
        $1:=Sous chaine($1;$vlPos+1)
    Sinon
        $2->{$vlElem}:=$1
    Fin de si
Jusque ($1="")
```

Référence

Position.

Supprimer chaine (source; position; nombreCar) → Alpha

Paramètre	Type		Description
source	Alpha	→	Chaîne de départ
position	Numérique	→	Premier caractère à supprimer
nombreCar	Numérique	→	Nombre de caractères à supprimer
Résultat	Alpha	←	Chaîne résultante

Description

Supprimer chaine supprime nombreCar dans source à partir de position et retourne la chaîne résultante.

Supprimer chaine retourne la même chaîne que source dans les cas suivants :
• source est une chaîne vide,
• position est supérieur à la longueur de source,
• nombreCar est égal à zéro (0).

Si position est inférieur à un (1), les caractères sont supprimés à partir du début de la chaîne.

Si position + nombreCar est supérieur ou égal à la longueur de source, les caractères sont supprimés à partir de position jusqu'à la fin de source.

Exemples

L'exemple suivant illustre l'utilisation de Supprimer chaine. Les résultats sont affectés à la variable vRésultat.

```
vRésultat := Supprimer chaine ("Lamborghini"; 6; 6)  ` vRésultat est égal à "Lambo"
vRésultat := Supprimer chaine ("Indentation"; 6; 2)  ` vRésultat est égal à "Indention"
     ` vRésultat est égal aux deux premiers caractères de var
vRésultat := Supprimer chaine (var; 3; 32000)
```

Référence

Inserer chaine, Remplacer caracteres, Remplacer chaine.

Trouver regex (motif; laChaîne{; début{; pos_trouvée; long_trouvée; *}}) → Chaîne

Paramètre	Type		Description
motif	Texte	→	Expression régulière
laChaîne	Texte	→	Chaîne dans laquelle s'effectue la recherche
début	Numérique	→	Position dans laChaîne où doit débuter la recherche
pos_trouvée	Var | Tab Entier long	←	Position de l'occurence
long_trouvée	Var | Tab Entier long	←	Longueur de l'occurence
*	*	→	Si passé : rechercher uniquement à la position indiquée
Résultat	Booléen	←	Vrai = la recherche a trouvé une occurrence, Faux sinon

Description

La commande Trouver regex permet de tester la conformité d'une chaîne de caractères par rapport à un ensemble de règles synthétisé au moyen d'un méta-langage appelé "expression régulière" ou "expression rationnelle". L'abrévation *regex* est communément employée pour désigner ces familles de notations.

Passez dans motif l'expression régulière à rechercher. Il s'agit d'une suite de caractères chargée de décrire une chaîne de caractères, à l'aide de caractères spéciaux.

Passez dans laChaîne la chaîne dans laquelle rechercher l'expression régulière.

Passez dans début la position dans laChaîne où doit débuter la recherche.

Si pos_trouvée et long_trouvée sont des variables, la commande retourne la position et la longueur de l'occurrence dans ces variables. Si vous passez des tableaux, la commande retourne la position et la longueur de l'occurrence dans l'élément zéro des tableaux et les positions et longueurs des groupes capturés par l'expression régulière dans les éléments suivants.

Le paramètre * indique, s'il est passé, que la recherche doit s'effectuer à la position définie par début sans chercher plus loin en cas d'échec.

La commande retourne Vrai si la recherche a trouvé une occurrence.

Pour plus d'informations sur les regex, reportez-vous par exemple à l'adresse suivante : http://fr.wikipedia.org/wiki/Expression_rationnelle

Pour plus d'informations sur la syntaxe de l'expression régulière passée dans le paramètre motif, reportez-vous à l'adresse suivante : http://www.icu-project.org/userguide/regexp.html

Exemples

Cette commande peut être utilisée de plusieurs manières, illustrées par les exemples ci-dessous :

(1) Recherche d'égalité complète :
vtrouvé:=Trouver regex(motif;montexte)

```
CHERCHER PAR FORMULE([Employés];Trouver regex(".*smith.*"; [Employés]nom))
```

(2) Recherche dans le texte par position :
vtrouvé:=Trouver regex(motif;montexte; début; pos_trouvée; long_trouvée)
Exemple pour afficher tous les tags de $1 :

```
début:=1
Repeter
    vtrouvé:=Trouver regex("<.*>";$1;début;pos_trouvée;long_trouvée)
    Si(vtrouvé)
        ALERTE(Sous chaine($1;pos_trouvée;long_trouvée))
        début:=pos_trouvée+long_trouvée
    Fin de si
Jusque(Non(vtrouvé))
```

(3) Recherche avec prise en charge des "groupes capturés" :
vtrouvé:=Trouver regex(motif;montexte; début; tab_pos_trouvée; tab_long_trouvée)

```
TABLEAU ENTIER LONG(tab_pos_trouvée;0)
TABLEAU ENTIER LONG(tab_long_trouvée;0)
vtrouvé:=Trouver regex("(.*)truc(.*)";$1;1;tab_pos_trouvée; tab_long_trouvée)
Si (vtrouvé)
    $group1:=Sous chaine($1;tab_pos_trouvée{1};tab_long_trouvée{1})
    $group2:=Sous chaine($1;tab_pos_trouvée{2};tab_long_trouvée{2})
Fin de si
```

(4) Recherche en limitant la comparaison de motif à la position indiquée :
Rajouter une étoile à la fin d'une des deux syntaxes précédentes.

```
vtrouvé:=Trouver regex("a.b";"---a-b---";1;$pos_trouvée;$long_trouvée)
    `retourne Vrai
vtrouvé:=Trouver regex("a.b";"---a-b---";1;$pos_trouvée;$long_trouvée;*)
    `retourne Faux
vtrouvé:=Trouver regex("a.b";"---a-b---";4;$pos_trouvée;$long_trouvée;*)
    `retourne Vrai
```

Note : Les positions et longueurs retournées n'ont de sens qu'en mode Unicode ou si le texte manipulé est de type ASCII 7 bits.

Gestion des erreurs

En cas d'erreur, la commande génére une erreur que vous pouvez intercepter via une méthode installée par la commande APPELER SUR ERREUR.

Windows vers Mac (texte) → Chaîne

Paramètre	Type		Description
texte	Chaîne	→	Texte en ANSI Windows
Résultat	Chaîne	←	Texte en ASCII Mac OS

Note de compatibilité : Cette commande fonctionne uniquement lorsque la base est exécutée en mode compatibilité ASCII. En mode Unicode, elle ne fait rien (la chaîne texte est retournée sans modification). A compter de la version 11 de 4D, cette commande est donc obsolète et son usage est déconseillé. Il est recommandé de convertir les chaînes de caractères à l'aide des commandes CONVERTIR DEPUIS TEXTE ou Convertir vers texte.

Description

Windows vers Mac retourne un texte exprimé avec la table ASCII Mac OS équivalent au texte passé dans texte, exprimé avec la table ANSI Windows.

Cette commande attend un paramètre de type Texte exprimé en ANSI Windows.

Normalement, vous n'avez pas besoin d'utiliser cette commande.

Dans 4D en mode compatibilité ASCII (Non Unicode), chaque valeur, champ ou variable de texte est encodée sur la base de la table ASCII Mac OS sous Macintosh et Windows, dans la mesure où vous ne les avez pas convertis à l'aide d'un filtre ASCII. Pour plus d'informations sur ce point, référez-vous à la section Codes ASCII.

Sous Windows, vous n'avez pas besoin de vous préoccuper de la conversion des codes ASCII. Lorsque vous copiez ou collez du texte entre 4D et Windows ou lorsque vous importez/exportez des données, 4D effectue automatiquement ces conversions. Cependant, lorsque vous utilisez des commandes de lecture/écriture telles que ENVOYER PAQUET ou RECEVOIR PAQUET, vous devez explicitement effectuer des conversions ASCII. C'est, en fait, le principal rôle de cette commande. Référez-vous à l'exemple ci-dessous.

Note : La commande remplace les caractères CRLF (Retour chariot + Retour à la ligne, codes de caractères 13 et 10) par des caractères CR simples. Par conséquent, le texte retourné peut être plus court que le texte d'origine.

Exemple

Lorsque vous lisez des caractères d'un document Windows à l'aide de RECEVOIR PAQUET, et si vous n'utilisez pas de filtre ASCII d'importation pour convertir les caractères Windows vers Mac OS (cf. la commande UTILISER FILTRE), il vous faut convertir vous-même le texte de Windows vers Mac OS. Vous pouvez le faire de la manière suivante :

```
` ...
RECEVOIR PAQUET ($vhDocRef;vtTexte;16*1024)
vtTexte:=Windows vers Mac(vtTexte)
` ...
```

Référence

Codes ASCII, Mac vers Windows, RECEVOIR PAQUET, UTILISER FILTRE.

6

Communications

ENVOYER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle envoyer l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

ENVOYER ENREGISTREMENT envoie l'enregistrement courant de table vers le port série ou vers un document ouvert par la commande REGLER SERIE. L'enregistrement est envoyé dans un format interne particulier ne pouvant être interprété que par la commande RECEVOIR ENREGISTREMENT. S'il n'y a pas d'enregistrement courant, ENVOYER ENREGISTREMENT ne fait rien.

L'enregistrement est envoyé en totalité, ce qui signifie que les images et les BLOBs stockés dans ou avec l'enregistrement sont également envoyés.

Important : Lorsque des enregistrements sont envoyés et reçus par ENVOYER ENREGISTREMENT et RECEVOIR ENREGISTREMENT, la structure de la table source et celle de la table de destination doivent être compatibles. Si ce n'est pas le cas, 4D convertira les valeurs en fonction des définitions des tables lorsque RECEVOIR ENREGISTREMENT sera exécutée.

Note : Si vous envoyez un enregistrement à un document avec cette commande, le document doit avoir été ouvert par la commande REGLER SERIE. Vous ne pouvez pas utiliser ENVOYER ENREGISTREMENT avec un document ouvert par Ouvrir document, Ajouter a document ou Creer document.

Exemple

Reportez-vous à l'exemple de la commande RECEVOIR ENREGISTREMENT.

Référence

ENVOYER VARIABLE, RECEVOIR ENREGISTREMENT, RECEVOIR VARIABLE.

ENVOYER PAQUET ({docRef; }paquet)

Paramètre	Type		Description
docRef	docRef	→	Référence de document ou canal courant (port série ou document)
paquet	Chaîne | BLOB	→	Chaîne ou BLOB à envoyer

Description

La commande ENVOYER PAQUET envoie paquet vers un port série ou un document. Si docRef est spécifié, le paquet est écrit dans le document référencé par docRef. Si docRef n'est pas spécifié, le paquet est envoyé vers le port série ou un document préalablement ouvert par la commande REGLER SERIE.

paquet représente une simple série de données, généralement une chaîne de caractères. Vous pouvez également passer un BLOB dans paquet. Ce principe permet notamment de s'affranchir des contraintes liées à l'encodage des caractères envoyés en mode texte (cf. exemple 2).

Note : Lorsque vous passez un BLOB dans paquet, la commande ne tient pas compte du jeu de caractères éventuellement défini par la commande UTILISER FILTRE. Le BLOB est envoyé sans aucune modification.

Avant d'utiliser ENVOYER PAQUET, vous devez ouvrir un port série ou un document avec la commande REGLER SERIE, ou un document avec une commande de gestion des documents.

Lorsque vous envoyez un paquet vers un document, le premier ENVOYER PAQUET commence à écrire les données au début du document — à moins que ce dernier n'ait été ouvert par la fonction Ajouter a document. Puis, jusqu'à ce que le document soit refermé, chaque paquet envoyé y est écrit à la suite du précédent.

Note : Ce fonctionnement est valide avec un document ouvert par REGLER SERIE. Cependant, pour un document ouvert par Ouvrir document, Creer document ou Ajouter a document, vous pouvez utiliser les commandes Position dans document et CHANGER POSITION DANS DOCUMENT pour connaître et modifier la position à laquelle, dans le document, la prochaine écriture (ENVOYER PAQUET) ou lecture (RECEVOIR PAQUET) aura lieu.

Important : En mode non Unicode (mode compatibilité), ENVOYER PAQUET envoie des caractères ASCII Mac sous Windows et sous Mac OS. Chacun d'entre eux est codé sur huit bits. Les caractères ASCII standard utilisent uniquement les sept bits inférieurs. De nombreux ordinateurs et périphériques n'utilisent pas le huitième bit de la même manière que Windows/Mac OS. Si la chaîne à envoyer contient des caractères utilisant le huitième bit, créez un filtre ASCII pour convertir les caractères ASCII, et exécutez UTILISER FILTRE avant d'utiliser ENVOYER PAQUET. Vous pouvez aussi utiliser la fonction Mac vers Windows (voir l'exemple de cette fonction).
Des protocoles tels que XON/XOFF utilisent certains codes ASCII inférieurs pour établir la communication entre les machines. Assurez-vous de ne pas envoyer de tels codes ASCII pour ne pas risquer d'interférer avec le protocole, voire de rompre la communication.

Exemples

(1) L'exemple suivant écrit, dans un document, des données en provenance de champs. Les valeurs sont écrites sous forme de champs de taille fixe. Dans ce cas, si la longueur d'un champ est inférieure à la taille fixée, le champ est comblé avec des espaces (c'est-à-dire que des espaces sont ajoutés de manière à ce que le champ corresponde à la taille définie). Bien que les champs de valeurs fixes soient un moyen peu efficace de stocker des données, certains systèmes informatiques et certaines applications l'utilisent encore :

```
$Doc := Creer document ("")    ` Création d'un document
Si (OK=1)    ` Est-ce que le document a bien été créé ?
    Boucle ($i; 1; Enregistrements trouves ([Personnes]))
            ` Boucle pour chaque enregistrement
        ENVOYER PAQUET ($Doc; Remplacer caracteres(15 * Caractere(Espacement);
                                                    [Personnes]Prénom;1))
            ` Envoi du paquet créé à partir d'une chaîne de 15 espaces contenant le
                                                                champ Prénom.
        ENVOYER PAQUET ($Doc; Remplacer caracteres(15 * Caractere(Espacement);
                                                        [Personnes]Nom;1))
            ` Envoi d'un second paquet créé à partir d'une chaîne de 15 espaces
            ` contenant le champ Nom. Cela aurait pu être mis dans le premier paquet,
            ` mais est séparé pour des raisons de clarté.
        ENREGISTREMENT SUIVANT([Personnes])
    Fin de boucle
    ENVOYER PAQUET ($Doc; Caractere (ASCII SUB))
        ` Envoi du code ASCII SUB, utilisé comme marqueur de fin d'enregistrement par
                                                            certains ordinateurs.
    FERMER DOCUMENT ($Doc)    ` Fermeture du document
Fin de si
```

(2) Cet exemple illustre l'envoi et la récupération de caractères étendus via un BLOB dans un document :

```
C_BLOB($blob_envoi)
C_BLOB($blob_reception)
TEXTE VERS BLOB("âzértÿ";$blob_envoi;UTF8 Texte sans longueur)
FIXER TAILLE BLOB($blob_envoi;16;255)
$blob_envoi{6}:=0
$blob_envoi{7}:=1
$blob_envoi{8}:=2
$blob_envoi{9}:=3
$blob_envoi{10}:=0
$vlDocRef:=Creer document("blob.test")
Si (OK=1)
    ENVOYER PAQUET($vlDocRef;$blob_envoi)
    FERMER DOCUMENT($vlDocRef)
Fin de si
$vlDocRef:=Ouvrir document(document)
    Si(OK=1)
    RECEVOIR PAQUET($vlDocRef;$blob_reception;65536)
    FERMER DOCUMENT($vlDocRef)
Fin de si
```

Référence

CHANGER POSITION DANS DOCUMENT, Position dans document, RECEVOIR PAQUET.

ENVOYER VARIABLE (variable)

Paramètre	Type		Description
variable	Variable	→	Variable à envoyer

Description

ENVOYER VARIABLE envoie variable vers le document ou le port série préalablement ouvert par la commande REGLER SERIE. La variable est envoyée dans un format interne spécial qui ne peut être relu que par la commande RECEVOIR VARIABLE. ENVOYER VARIABLE envoie la totalité de la variable (y compris son type et sa valeur).

Notes

1. Si vous envoyez une variable à un document avec cette commande, le document doit avoir été ouvert par la commande REGLER SERIE. Vous ne pouvez pas utiliser ENVOYER VARIABLE avec un document ouvert par Ouvrir document, Ajouter a document ou Creer document.
2. Cette commande ne fonctionne pas avec les variables de type tableau. Si vous voulez envoyer et recevoir des tableaux via un document ou un port série, utilisez les nouvelles Commandes du thème BLOB, apparues avec la version 6 de 4D.

Exemple

Reportez-vous à l'exemple de la commande RECEVOIR ENREGISTREMENT.

Référence

ENVOYER ENREGISTREMENT, RECEVOIR ENREGISTREMENT, RECEVOIR VARIABLE, REGLER SERIE.

FIXER TIMEOUT (secondes)

Paramètre	Type	Description
secondes	Numérique →	Nombre de secondes jusqu'au timeout

Description

La commande FIXER TIMEOUT vous permet de définir le temps d'attente maximum pour l'exécution d'une commande de communication série. Si la commande ne se termine pas dans le temps secondes qui lui est imparti, la communication série est annulée, l'erreur –9990 est générée, et la variable système OK prend la valeur 0. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

Notez que le délai défini représente la durée totale permise pour que la commande s'exécute, et non le délai d'attente entre chaque caractère reçu. Pour annuler un paramétrage précédent et ne pas spécifier de temps d'attente maximum, passez 0 dans le paramètre secondes.

Les commandes de communication série affectées par ce paramétrage sont les suivantes :

- RECEVOIR PAQUET
- RECEVOIR ENREGISTREMENT
- RECEVOIR VARIABLE

Exemples

L'exemple suivant fixe le port série devant recevoir des données et le timeout. Les données sont lues à l'aide de RECEVOIR PAQUET. Si les données ne sont pas bien reçues dans le temps défini, une erreur survient :

```
    ` Ouverture du port série
REGLER SERIE (Port série MacOS; Vitesse 9600 + Bits de données 8 + Bit de stop un +
                                                        Pas de parité)
FIXER TIMEOUT (10)  ` Fixer le timeout à 10 secondes
    ` Traiter les interruptions éventuelles
APPELER SUR ERREUR ("INTERCEPTER ERREURS COMMUNICATIONS")
RECEVOIR PAQUET (vBuffer; Caractere (Retour chariot))  ` Lire jusqu'au retour chariot
```

```
Si (OK = 0)  ` Si une erreur survient
    ALERTE ("Erreur lors de la réception des données.")  ` Informer l'utilisateur
Sinon
    [Personnes]Nom := vBuffer  ` Sauvegarder les données dans un champ
Fin de si
```

Référence

APPELER SUR ERREUR, RECEVOIR BUFFER, RECEVOIR ENREGISTREMENT, RECEVOIR PAQUET, RECEVOIR VARIABLE.

LIRE CORRESPONDANCE PORT SERIE (tabNums; tabNoms)

Paramètre	Type		Description
tabNums	Tab numérique	←	Tableau de numéros de ports série
tabNoms	Tableau chaîne	←	Tableau de noms de ports série

Description

La commande LIRE CORRESPONDANCE PORT SERIE retourne deux tableaux tabNums et tabNoms contenant respectivement la liste des numéros et des noms des ports série de la machine courante.

Cette commande est utile sous Mac OS X car le système alloue dynamiquement les numéros des ports série lorsque vous utilisez un adaptateur série USB. A l'aide de cette commande, vous pouvez adresser les ports série étendus via leur nom (invariable), quel que soit leur numéro.

Note : Cette commande ne retourne pas de valeurs significatives avec les ports standard. Si vous souhaitez adresser un port standard, vous devez passer directement sa valeur (0 ou 1) à la commande REGLER SERIE (ancien mode de fonctionnement de 4D).

Exemple

Cette méthode projet permet d'adresser le même port série (sans protocole), quel que soit le numéro qui lui a été attribué :

```
TABLEAU TEXTE ($tNomPorts;0)
TABLEAU ENTIER LONG ($tNumPorts;0)
C_ENTIER LONG($vNumport;$vNumportFinal)

    `Connaître les numéros actuels des ports série
LIRE CORRESPONDANCE PORT SERIE($tNumPorts;$tNomPorts)
$vNumport:=Chercher dans tableau($tNomPorts;vNomport)
    ` vNomport contient le nom du port à utiliser, il peut provenir d'une boîte de
    ` dialogue, d'une valeur stockée dans un champ, etc.
Si(tNumPorts{$vNumport}=0)
    $vNumportFinal:=0   `cas particulier sous Mac OS X
Sinon
    $vNumportFinal:=tNumPorts{$vNumport}+100
Fin de si
```

```
    `params contient les paramètres de communication
REGLER SERIE($vNumportFinal;params)
... `Effectuer ici les opérations souhaitées
REGLER SERIE (11)     `Fermeture du port
```

Référence

REGLER SERIE.

RECEVOIR BUFFER (varRéception)

Paramètre	Type		Description
varRéception	Variable	→	Variable devant recevoir les données

Description

La commande RECEVOIR BUFFER lit les données du port série préalablement ouvert par la commande REGLER SERIE. Le port série comporte un buffer qui se remplit de caractères jusqu'à ce qu'une commande les charge. RECEVOIR BUFFER récupère les caractères présents dans le buffer, les place dans la variable varRéception puis vide le buffer. S'il n'y a pas de caractères dans le buffer, la variable varRéception est vide.

Sous Windows
Le buffer du port série sous Windows a une capacité limitée à 10 Ko. Cela signifie que le buffer peut être saturé. Lorsqu'il est plein et que de nouveaux caractères sont reçus, ils remplacent les plus anciens caractères. Les anciens caractères sont perdus ; par conséquent, il est important que le buffer soit lu rapidement lorsque des nouvelles données sont reçues.

Sous Mac OS
Le buffer du port série sous Mac OS 9.x a une capacité limitée à 10 Ko. Sous Mac OS X, sa capacité est en principe illimitée (elle dépend de la mémoire disponible). Si le buffer est saturé et que de nouveaux caractères sont reçus, ils remplacent les plus anciens caractères. Les anciens caractères sont perdus ; par conséquent, il est important que le buffer soit lu rapidement lorsque des nouvelles données sont reçues.

Note : Il existe des plug-ins 4D qui vous permettent d'augmenter la capacité du port série.

La commande RECEVOIR BUFFER est différente de RECEVOIR PAQUET dans la mesure où elle récupère tout ce qui se trouve dans le buffer et le retourne immédiatement. RECEVOIR PAQUET, pour sa part, attend de récupérer un caractère spécifique ou un certain nombre de caractères.

Pendant l'exécution d'un RECEVOIR BUFFER, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur –9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

Exemple

La méthode projet ECOUTER PORT SÉRIE utilise RECEVOIR BUFFER pour récupérer du texte depuis le port série et l'accumuler dans une variable interprocess :

```
    ` ECOUTER PORT SÉRIE
    ` Ouvrir le port série
REGLER SERIE (201; Vitesse 9600 + Bits de données 8 + Bit de stop un + Pas de parité)
<>IP_Ecouter_Port_Série:=Vrai
Tant que (<>IP_Ecouter_Port_Série)
    RECEVOIR BUFFER($vtBuffer)
    Si ((Longueur($vtBuffer)+Longueur(<>vtBuffer))>MAXLARGTEXTE)
        <>vtBuffer:=""
    Fin de si
    <>vtBuffer:=<>vtBuffer+$Buffer
Fin tant que
```

A ce stade, tout autre process peut lire la variable interprocess <>vtBuffer pour exploiter les données en provenance du port série.

Pour cesser d'écouter le port série, exécutez simplement la méthode suivante :

```
    ` Fin de l'écoute du port série
<>IP_Ecouter_Port_Série:=Faux
```

Notez que l'accès à la variable interprocess <>vtBuffer doit être protégé par un sémaphore, de manière à ce que les process n'entrent pas en conflit (reportez-vous à la description de la fonction Semaphore pour plus d'informations).

Référence

APPELER SUR ERREUR, RECEVOIR PAQUET, REGLER SERIE, Semaphore, Variables.

RECEVOIR ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table dans laquelle recevoir l'enregistrement, ou Table par défaut si omis

Description

RECEVOIR ENREGISTREMENT ajoute dans table un enregistrement reçu par l'intermédiaire du port série ou d'un document ouvert par la commande REGLER SERIE. L'enregistrement doit avoir été envoyé par la commande ENVOYER ENREGISTREMENT. Lorsque vous exécutez RECEVOIR ENREGISTREMENT, un nouvel enregistrement est automatiquement créé dans table. Si l'enregistrement a été correctement reçu, vous pouvez le sauvegarder à l'aide de STOCKER ENREGISTREMENT.

L'enregistrement est reçu en totalité, ce qui signifie que les images et BLOBs stockés dans ou avec l'enregistrement sont également reçus.

Important : Lorsque des enregistrements sont envoyés et reçus par ENVOYER ENREGISTREMENT et RECEVOIR ENREGISTREMENT, la structure de la table source et celle de la table de destination doivent être compatibles. Si ce n'est pas le cas, 4D convertira les valeurs en fonction des définitions des tables lorsque RECEVOIR ENREGISTREMENT sera exécutée.

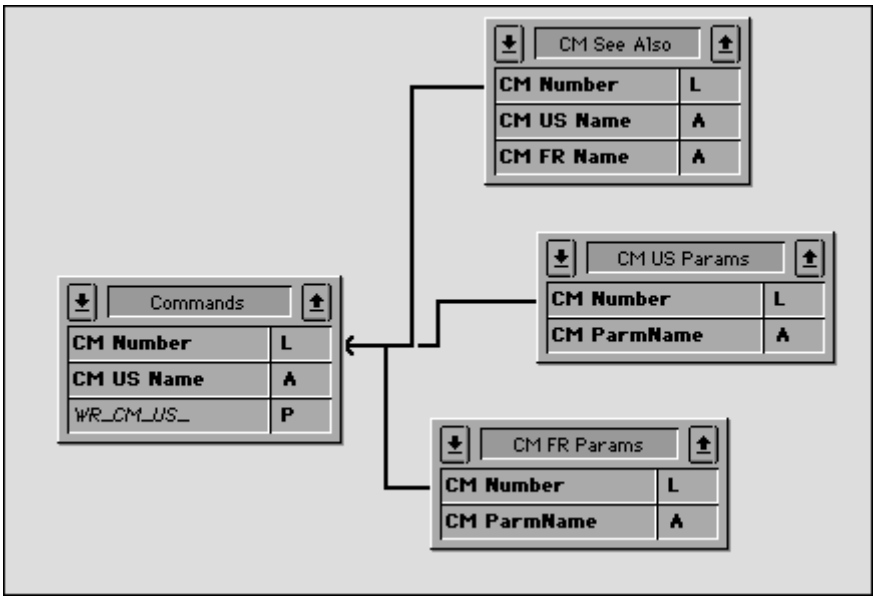
Notes

1. Si vous recevez un enregistrement provenant d'un document avec cette commande, le document doit avoir été ouvert par la commande REGLER SERIE. Vous ne pouvez pas utiliser RECEVOIR ENREGISTREMENT avec un document ouvert par Ouvrir document, Ajouter a document ou Creer document.
2. Pendant l'exécution d'un RECEVOIR ENREGISTREMENT, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur –9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande APPELER SUR ERREUR. Généralement, vous devez gérer les interruptions d'une réception uniquement lors d'une communication série.

Exemple

L'utilisation combinée de ENVOYER VARIABLE, ENVOYER ENREGISTREMENT, RECEVOIR VARIABLE et RECEVOIR ENREGISTREMENT est idéale pour archiver des données ou échanger des données entre des bases monopostes identiques utilisées à différents endroits. Certes, vous pouvez échanger des données entre des bases 4D à l'aide des commandes d'import/export telles que IMPORTER TEXTE et EXPORTER TEXTE. Cependant, si vos données contiennent des images, des sous-tables et/ou des tables liées, l'utilisation de ENVOYER ENREGISTREMENT et RECEVOIR ENREGISTREMENT est, de loin, plus pratique.

Par exemple, la documentation que vous êtes en train de lire a été créée à l'aide de 4D et 4D Write. Comme plusieurs rédacteurs basés dans différents pays travaillaient sur ce projet, nous avions besoin d'un système simple pour échanger les données entre les différentes bases. Voici une vue simplifiée de la structure de la base :



La table [Commands] contient la description de chaque commande ou section. Les tables [CM US Params] et [CM FR Params] contiennent respectivement les paramètres de chaque commande en anglais et en français. La table [CM See Also] contient les commandes indiquées en tant que Références pour chaque commande ou section. L'échange de la documentation entre les bases consiste donc à envoyer les enregistrements de [Commands] ainsi que leurs enregistrements liés. Pour cela, nous utilisons ENVOYER ENREGISTREMENT et RECEVOIR ENREGISTREMENT. De plus, nous utilisons ENVOYER VARIABLE et RECEVOIR VARIABLE pour "cocher" les enregistrements importés/exportés.

Voici la méthode projet (simplifiée) d'export de la documentation :

```
    ` Méthode projet CM_EXPORT_SEL
    ` Cette méthode fonctionne avec la sélection courante de la table [Commands]

REGLER SERIE(12;"") ` Laissons l'utilisateur créer et ouvrir un document série
Si (OK=1)
        ` Marquons le document avec une variable décrivant son contenu
        ` Note: la variable process BUILD_LANG indique si des données US (anglaises)
        ` ou FR (françaises) sont envoyées
    $vsTag:="4DV6COMMAND"+BUILD_LANG
    ENVOYER VARIABLE($vsTag)
        ` Envoyer une variable indiquant combien de [Commands] sont exportées
    $vlNbCmd:=Enregistrements trouves([Commands])
    ENVOYER VARIABLE($vlNbCmd)
    DEBUT SELECTION([Commands])
        ` Pour chaque commande
    Boucle ($vlCmd;1;$vlNbCmd)
            ` Envoyer l'enregistrement [Commands]
        ENVOYER ENREGISTREMENT([Commands])
            ` Sélection de tous les enregistrements liés
        LIEN RETOUR([Commands])
            ` En fonction de la langue, envoyer une variable indiquant
            ` le nombre de paramètres qui va suivre
        Au cas ou
            : (BUILD_LANG="US")
                $vlNbParm:=Enregistrements trouves([CM US Params])
            : (BUILD_LANG="FR")
                $vlNbParm:=Enregistrements trouves([CM FR Params])
        Fin de cas
```

```
            ENVOYER VARIABLE($vlNbParm)
                ` Envoyer les enregistrements des paramètres (s'il y en a)
            Boucle ($vlParm;1;$vlNbParm)
                Au cas ou
                    : (BUILD_LANG="US")
                        ENVOYER ENREGISTREMENT([CM US Params])
                        ENREGISTREMENT SUIVANT([CM US Params])
                    : (BUILD_LANG="FR")
                        ENVOYER ENREGISTREMENT([CM FR Params])
                        ENREGISTREMENT SUIVANT([CM FR Params])
                 Fin de cas
            Fin de boucle
                ` Envoyer une variable indiquant combien de "Références" vont suivre
            $vlNbSee:=Enregistrements trouves([CM See Also])
            ENVOYER VARIABLE($vlNbSee)
                ` Envoyer les enregistrements [See Also] (s'il y en a)
            Boucle ($vlSee;1;$vlNbSee)
                ENVOYER ENREGISTREMENT([CM See Also])
                ENREGISTREMENT SUIVANT([CM See Also])
            Fin de boucle
                ` Aller à l'enregistrement [Commands] suivant et continuer l'export
            ENREGISTREMENT SUIVANT([Commands])
        Fin de boucle
        REGLER SERIE(11) ` Fermer le document
    Fin de si
```

Voici la méthode projet (simplifiée) d'import de la documentation :

```
    ` Méthode projet CM_IMPORT_SEL

REGLER SERIE(10;"")   ` Laissons l'utilisateur ouvrir un document existant
Si (OK=1) ` Si un document a été ouvert
    RECEVOIR VARIABLE($vsTag)   ` Essayons de recevoir la variable marqueur attendue
    Si ($vsTag="4DV6COMMAND@")   ` Avons-nous le bon marqueur ?
            ` Extrayons la langue du marqueur
        $CurLang:=Sous chaine($vsTag;Longueur($vsTag)-1)
        Si (($CurLang="US") | ($CurLang="FR"))   ` Avons-nous reçu un langage valide ?
                ` Combien de commandes dans ce document ?
            RECEVOIR VARIABLE($vlNbCmd)
```

```
            Si ($vlNbCmd>0)   ` S'il en existe une au moins
                    ` Pour chaque enregistrement [Commands] archivé
                Boucle ($vlCmd;1;$vlNbCmd)
                        ` Réception de l'enregistrement
                    RECEVOIR ENREGISTREMENT([Commands])
                        ` Appelons une sous-routine qui sauvegarde le
                        ` nouvel enregistrement ou le copie
                        ` dans un enregistrement existant
                    CM_IMP_CMD ($CurLang)
                        ` Réception du nombre de paramètres (s'il y en a)
                    RECEVOIR VARIABLE($vlNbParm)
                    Si ($vlNbParm>=0)
                            ` Appelons une sous-routine qui appelle
                            ` RECEVOIR ENREGISTREMENT puis stocke
                            ` les nouveaux enregistrements ou les copie dans des
                            ` enregistrements existants
                        CM_IMP_PARM ($vlNbParm;$CurLang)
                    Fin de si
                        ` Réception du nombre de "Références" (s'il y en a)
                    RECEVOIR VARIABLE($vlNbSee)
                    Si ($vlNbSee>0)
                            ` Appelons une sous-routine qui appelle
                            ` RECEVOIR ENREGISTREMENT puis stocke
                            ` les nouveaux enregistrements ou les copie dans des
                            ` enregistrements existants
                        CM_IMP_SEEA ($vlNbSee;$CurLang)
                    Fin de si
                Fin de boucle
            Sinon
                ALERTE("Le nombre de commandes dans ce document d'export est
                                                                    invalide.")
            Fin de si
        Sinon
            ALERTE("Le langage de ce document d'export est inconnu.")
        Fin de si
    Sinon
        ALERTE("Ce document n'est pas un document d'export.")
    Fin de si
    REGLER SERIE(11)   ` Fermer document
Fin de si
```

Notez que nous n'avons pas testé la variable OK pendant la réception des données, si intercepté les éventuelles erreurs. Cependant, comme nous avons stocké dans le document des variables décrivant le document lui-même, si ces variables, une fois reçues, sont correctes, la probabilité d'erreur est très faible. Si par exemple un utilisateur ouvre un mauvais document, le premier test stoppe l'opération entière.

Référence

ENVOYER ENREGISTREMENT, ENVOYER VARIABLE, RECEVOIR VARIABLE.

Variables et ensembles système

La variable système OK prend la valeur 1 si l'enregistrement est correctement reçu, sinon elle prend la valeur 0.

RECEVOIR PAQUET ({docRef; }réceptVar; stopCar | nbCar)

Paramètre	Type		Description
docRef	docRef	→	Numéro de référence de document ou canal courant (port série ou document)
réceptVar	Var chaîne |Var BLOB	→	Variable devant recevoir les données
stopCar | nbCar	Alpha | Num	→	Caractère(s) au(x)quel(s) stopper la réception des données ou Nombre de caractères à recevoir

Description

La commande RECEVOIR PAQUET lit des caractères depuis un port série ou un document.

Si docRef est spécifié, la commande récupère des caractères depuis un document ouvert par la fonction Ouvrir document, Creer document ou Ajouter a document. Si docRef est omis, la commande récupère des caractères depuis un port série ou un document ouvert par la commande REGLER SERIE.

Dans tous les cas, les caractères lus sont retournés dans la variable réceptVar, qui doit être une variable de type Texte, Alpha ou BLOB. Si les caractères ont été envoyés par la commande ENVOYER PAQUET, le type doit correspondre à celui du paquet envoyé.

Notes :

• En mode non Unicode (mode compatibilité), les variables alpha acceptent jusqu'à 255 caractères et ont une taille fixe, les variables texte n'ont pas de taille prédéfinie et acceptent jusqu'à 32000 caractères.
•Si le paquet reçu est de type BLOB, la commande ne tient pas compte du jeu de caractères éventuellement défini par la commande UTILISER FILTRE. Le BLOB est retourné sans aucune modification.

Si vous voulez recevoir un nombre prédéfini de caractères, passez ce nombre dans le paramètre nbCar. Si la variable réceptVar est de type Texte, vous pouvez lire en un seul appel jusqu'à 2 Go de texte en mode Unicode ou 32000 caractères en mode non Unicode (dans ce cas, pour spécifier le nombre maximum de caractères, vous pouvez passer la constante MAXLONGTEXTEAVANTV11 dans nbCar).

Si vous voulez recevoir des caractères jusqu'à ce qu'une chaîne de caractères (comportant un ou plusieurs caractères) soit lue, passez-la dans le paramètre stopCar (la chaîne n'est pas retournée dans réceptVar).
Dans ce cas, si la chaîne de caractères spécifiée par stopCar n'est pas trouvée :
• lorsque RECEVOIR PAQUET lit un document, l'exécution de la commande se terminera à la fin du document.
• lorsque RECEVOIR PAQUET lit des données en provenance du port série, la commande s'exécutera indéfiniment jusqu'à ce que le délai d'attente (s'il est fixé) soit écoulé (cf. la commande FIXER TIMEOUT) ou que l'utilisateur interrompe la réception (voir ci-dessous).

Pendant l'exécution d'un RECEVOIR PAQUET, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur –9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande APPELER SUR ERREUR. Généralement, vous devez gérer les interruptions d'une réception uniquement lors d'une communication série.

Lors de la lecture d'un document, le premier RECEVOIR PAQUET commence par lire le début du document. La lecture des paquets suivants débute au caractère situé immédiatement après le dernier caractère lu.

Note : Ce fonctionnement est valide avec un document ouvert par REGLER SERIE. Cependant, pour un document ouvert par Ouvrir document, Creer document ou Ajouter a document, vous pouvez aussi utiliser les commandes Position dans document et CHANGER POSITION DANS DOCUMENT pour connaître et modifier la position à laquelle, dans le document, la prochaine écriture (ENVOYER PAQUET) ou lecture (RECEVOIR PAQUET) aura lieu.

En cas de tentative de lecture après la fin d'un document, RECEVOIR PAQUET retourne les données lues jusqu'à ce point et la variable système OK prend la valeur 1. Les RECEVOIR PAQUET suivants retourneront une chaîne vide et OK prendra la valeur zéro.

Note : En mode non Unicode (compatibilité), lorsque vous lisez des caractères d'un document Windows avec RECEVOIR PAQUET, si vous n'utilisez pas de filtre ASCII d'importation pour convertir les caractères Windows vers Mac OS (cf. la commande UTILISER FILTRE), vous pouvez utiliser la fonction Windows vers Mac (voir l'exemple de cette fonction).

Exemples

(1) L'exemple suivant lit 20 caractères depuis un port série et les place dans la variable RécupVingt :

```
RECEVOIR PAQUET (RécupVingt; 20)
```

(2) L'exemple suivant lit des données depuis le document référencé par la variable MonDoc et les place dans la variable vData. La commande récupère les données jusqu'à ce qu'elle rencontre un retour chariot :

```
RECEVOIR PAQUET (MonDoc; vData; Caractere (Retour chariot))
```

3) L'exemple suivant lit des données du document référencé par la variable MonDoc et les place dans la variable vData. La commande récupère les données jusqu'à ce qu'elle rencontre une balise HTML de fin de tableau (</TD>) :

```
RECEVOIR PAQUET (MonDoc; vData; "</TD>")
```

(4) L'exemple suivant lit des données d'un document et les place dans des champs. Les données sont stockées dans des champs de longueur fixe. La méthode fait appel à une sous-routine pour éliminer les espaces superflus (situés derrière les valeurs). Le code de la sous-routine est présenté après la méthode :

```
$Doc := Ouvrir document ("";"TEXT")  ` Ouverture d'un document de type Texte
Si (OK=1)  ` Si le document est ouvert…
   Repeter  ` Boucle jusqu'à ce qu'il n'y ait plus de données
      RECEVOIR PAQUET ($Doc; $Var1; 15)  ` Lecture de 15 caractères
      RECEVOIR PAQUET ($Doc; $Var2; 15)  ` Même chose pour le second champ
      Si (OK = 1) ` Si ce n'est pas la fin du document…
         CREER ENREGISTREMENT([Personnes])  ` Créer un nouvel enregistrement
         [Personnes]Prénom := Elimine ($Var1)  ` Sauvegarder le prénom
         [Personnes]Nom := Elimine ($Var2)  ` Sauvegarder le nom
         STOCKER ENREGISTREMENT([Personnes])  ` Sauvegarder l'enregistrement
      Fin de si
   Jusque (OK =0)
   FERMER DOCUMENT ($Doc)  ` Fermeture du document
Fin de si
```

Les espaces superflus derrière les valeurs sont éliminés par la méthode suivante, appelée Elimine :

```
Boucle ($i; Longueur ($1); 1; -1)  ` Boucle sur la fin de la chaîne d'où démarrer
   Si ($1[[$i]] # " ")  ` Si ce n'est pas un espace…
      $i := -$i  ` Forcer la boucle à stopper
   Fin de si
Fin de boucle
$0 := Supprimer chaine($1; -$i;Longueur($1))  ` Suppression des espaces
```

Référence

CHANGER POSITION DANS DOCUMENT, ENVOYER PAQUET, FIXER TIMEOUT, Position dans document, RECEVOIR PAQUET, UTILISER FILTRE.

Variables et ensembles système

Après un appel à RECEVOIR PAQUET, la variable système OK prend la valeur 1 si le paquet est reçu sans erreur. Sinon, OK prend la valeur 0.

RECEVOIR VARIABLE (variable)

Paramètre	Type		Description
variable	Variable	→	Variable dans laquelle recevoir une variable

Description

La commande RECEVOIR VARIABLE reçoit variable, une variable envoyée par la commande ENVOYER VARIABLE, depuis un document ou un port série préalablement ouvert par la commande REGLER SERIE.

En mode interprété, si la variable n'existe pas préalablement à l'appel de RECEVOIR VARIABLE, elle sera créée, typée et remplie en fonction de ce qui a été reçu. En mode compilé, la variable doit être du même type que celle qui est reçue. Dans les deux cas, le contenu de la variable est remplacé par celui de la variable reçue.

Notes

1. Si vous recevez une variable depuis un document avec cette commande, le document doit avoir été ouvert par la commande REGLER SERIE. Vous ne pouvez pas utiliser RECEVOIR VARIABLE avec un document ouvert par Ouvrir document, Ajouter a document ou Creer document.
2. Cette commande ne fonctionne pas avec les variables de type tableau. Si vous voulez envoyer et recevoir des tableaux via un document ou un port série, utilisez les nouvelles Commandes du thème BLOB, apparues avec la version 6 de 4D.
3. Pendant l'exécution d'un RECEVOIR VARIABLE, l'utilisateur peut interrompre l'opération en appuyant sur les touches **Ctrl**+**Alt**+**Maj** (sous Windows) ou **Commande**+**Option**+**Maj** (sous Mac OS). Cette interruption génère une erreur –9994 que vous pouvez intercepter à l'aide d'une méthode installée par la commande APPELER SUR ERREUR. Généralement, vous devez gérer les interruptions d'une réception uniquement lors d'une communication série.

Exemple

Reportez-vous à l'exemple de la commande RECEVOIR ENREGISTREMENT.

Référence

APPELER SUR ERREUR, ENVOYER ENREGISTREMENT, ENVOYER VARIABLE, RECEVOIR ENREGISTREMENT.

Variables et ensembles système

La variable système OK prend la valeur 1 si la variable est correctement reçue, sinon elle prend la valeur 0.

REGLER SERIE (port | opération{; param | document})

Paramètre	Type	Description
port | opération	Numérique →	Numéro de port série ou opération à effectuer sur document
param | document	Num | Alpha→	Paramètres de communication ou Nom du document

Description

La commande REGLER SERIE permet d'ouvrir un port série ou un document. Vous ne pouvez ouvrir qu'un port série ou un document à la fois avec cette commande.

Note historique : A l'origine, REGLER SERIE a été la première commande 4D permettant de travailler avec les ports série et des documents sur disque. Depuis, de nouvelles commandes ont été ajoutées. Aujourd'hui, vous pouvez généralement travailler avec des documents sur disque à l'aide des commandes Ouvrir document, Creer document et Ajouter a document, puis lire et écrire des caractères dans les documents avec RECEVOIR PAQUET et ENVOYER PAQUET (ces deux commandes fonctionnent aussi avec REGLER SERIE). Cependant, si vous souhaitez utiliser les commandes ENVOYER VARIABLE, RECEVOIR VARIABLE, ENVOYER ENREGISTREMENT et RECEVOIR ENREGISTREMENT, vous devez appeler REGLER SERIE pour accéder aux documents sur disque.

La description de la commande REGLER SERIE se compose de deux sections :
• Travailler avec les ports série
• Travailler avec des documents

Travailler avec les ports série : REGLER SERIE(port;param)

La première syntaxe de REGLER SERIE ouvre un port série et définit le protocole de communication ainsi que des informations supplémentaires. Les données peuvent être envoyées par les commandes ENVOYER PAQUET, ENVOYER ENREGISTREMENT ou ENVOYER VARIABLE, et reçues par les commandes RECEVOIR BUFFER, RECEVOIR PAQUET, RECEVOIR ENREGISTREMENT ou RECEVOIR VARIABLE.

• Le premier paramètre, port, définit le port et le protocole utilisés. Vous pouvez adresser jusqu'à 99 ports série (un par un).
Le tableau suivant liste les valeurs possibles du paramètre port :

Valeurs port	Description
0	Port imprimante (Mac) ou COM2 (Windows) sans protocole
1	Port modem (Mac) ou COM1 (Windows) sans protocole
20	Port imprimante (Mac) ou COM2 (Windows) avec protocole logiciel tel que XON/XOFF
21	Port modem (Mac) ou COM1 (Windows) avec protocole logiciel tel que XON/XOFF
30	Port imprimante (Mac) ou COM2 (Windows) avec protocole matériel tel que RTS/CTS)
31	Port modem (Mac) ou COM1 (Windows) avec protocole matériel tel que RTS/CTS)
101 à 199	Communication série sans protocole*
201 à 299	Communication série avec protocole logiciel tel que XON/XOFF*
301 à 399	Communication série avec protocole matériel tel que RTS/CTS*

*** Important** : La valeur que vous passez dans port doit désigner un port série "logique" reconnu par votre système d'exploitation. Par exemple, pour que vous puissiez utiliser les valeurs 101, 203 et 325, les ports série COM1, COM3 et COM25 doivent avoir été correctement configurés.

Note sur les ports série
En standard, les systèmes Mac OS et Windows reconnaissent deux ports série logiques : sous Mac OS, le port modem et le port imprimante ; sous Windows, les ports COM1 et COM2. Toutefois, des ports série supplémentaires peuvent être ajoutés, par l'intermédiaire de cartes d'extension. 4D n'adressait à l'origine que les deux ports série standard, et a intégré par la suite la gestion des ports série supplémentaires. Pour des raisons de compatibilité, les deux systèmes d'adressage ont été conservés.
- Si vous souhaitez adresser uniquement un port série standard (imprimante/COM2 ou modem/COM1), vous pouvez passer dans le paramètre port soit une des valeurs 0, 1, 20, 21, 30 et 31 (correspondant à l'ancien mode de fonctionnement de 4D), soit une valeur > 100 (cf. ci-dessous).
- Si vous souhaitez adresser des ports série "étendus", vous devez passer dans port (pour adresser le N^{ième} port série) la valeur N+100, augmentée éventuellement de 100 ou de 200, si vous voulez utiliser respectivement un protocole logiciel ou matériel.

Exemples :

(1) Vous souhaitez utiliser le port imprimante/COM2 sans protocole, vous pouvez utiliser l'une des syntaxes suivantes :

REGLER SERIE (0;param)

ou

REGLER SERIE (102;param)

(2) Vous souhaitez utiliser le port modem/COM1 avec le protocole XON/XOFF, vous pouvez utiliser l'une des syntaxes suivantes :

REGLER SERIE (21;param)

ou

REGLER SERIE (201;param)

(3) Vous souhaitez utiliser le port COM25 avec le protocole RTS/CTS, vous devez utiliser la syntaxe suivante :

REGLER SERIE (325;param)

• Le second paramètre, param, permet de fixer la vitesse, le nombre de bits de données, le nombre de bits de stop et la parité. La valeur de param se calcule en additionnant les valeurs de vitesse, de bits de données, de bits de stop et de parité, telles que définies dans le tableau ci-dessous. Par exemple, pour paramétrer la communication à 1200 bauds, 8 bits de données, 1 bit de stop et aucune parité, passez 19550 (soit 94+3072+16384+0) dans le paramètre param.

Contrôle	Valeur param (à cumuler)	Fonction
Vitesse (en bauds)	380	300
	189	600
	94	1200
	62	1800
	46	2400
	30	3600
	22	4800
	14	7200
	10	9600
	4	19200
	2	28800
	1	38400
	0	57600
	1022	115200
	1021	230400

Bits de données	0	5
	2048	6
	1024	7
	3072	8
Bits de stop	16384	1
	–32768	1,5
	–16384	2
Parité	0	Aucune
	4096	Impaire
	12288	Paire

Astuce : Les différentes valeurs numérique à cumuler et à passer dans les paramètres port et param (à l'exception des valeurs de COM1...COM99) sont disponibles en tant que **Constantes** prédéfinies dans le thème **Communications** de l'Explorateur, en mode Développement. Pour les valeurs de COM1...COM99, vous devez utiliser des valeurs numériques littérales.

Lorsque vous n'avez plus besoin d'un port série, vous devez le refermer. Pour cela, appelez de nouveau REGLER SERIE et passez-lui la valeur 11. Exemple :

```
REGLER SERIE(11)  `Referme un port série préalablement ouvert
```

Travailler avec des documents : REGLER SERIE(opération;document)

La seconde syntaxe de la commande REGLER SERIE vous permet de créer, ouvrir ou fermer un document. A la différence des commandes du thème Documents système, REGLER SERIE ne permet d'ouvrir qu'un document à la fois. Le document peut être "lu à partir de" ou "écrit dans". Reportez-vous à la section Présentation des documents système pour plus d'informations sur ce point.

Le premier paramètre, opération, définit l'opération à effectuer avec le document désigné par document. Le tableau suivant dresse la liste les valeurs d'opération et le résultat obtenu, en fonction de la valeur de document.
La première colonne fournit les valeurs possibles du paramètre opération. La deuxième colonne fournit les valeurs possibles du paramètre document. La troisième colonne décrit le résultat obtenu. Par exemple, pour afficher un fichier de type texte dans une boîte de dialogue standard d'ouverture de document, vous pouvez écrire l'instruction suivante :

```
REGLER SERIE (13; "")
```

Opération	Document	Résultat
10	Chaîne	Ouvre le document dont le nom est spécifié par Chaîne Si le document n'existe pas, il est créé et ouvert.
10	"" (chaîne vide)	Affiche la boîte de dialogue d'ouverture de fichier. Tous les types de fichiers sont présentés.
11	Aucun	Referme un fichier ouvert.
12	"" (chaîne vide)	Affiche la boîte de dialogue standard d'enregistement de fichier, permettant de créer un nouveau fichier.
13	"" (chaîne vide)	Affiche la boîte de dialogue d'ouverture de fichier. Seuls les fichiers de type Texte sont présentés.

Toutes les opérations décrites dans ce tableau modifient la variable système Document en conséquence. De plus, la variable système OK prend la valeur 1 si l'opération s'est déroulée correctement, 0 sinon.

Exemples

Reportez-vous aux exemples des commandes RECEVOIR BUFFER, FIXER TIMEOUT et RECEVOIR ENREGISTREMENT.

Référence

Ajouter a document, Creer document, ENVOYER ENREGISTREMENT, ENVOYER PAQUET, ENVOYER VARIABLE, FIXER TIMEOUT, LIRE CORRESPONDANCE PORT SERIE, Ouvrir document, RECEVOIR BUFFER, RECEVOIR ENREGISTREMENT, RECEVOIR PAQUET, RECEVOIR VARIABLE.

UTILISER FILTRE (filtre | *{; typeFiltre})

Paramètre	Type		Description
filtre | *	Alpha | *	→	Nom du jeu de caractères à utiliser (mode Unicode) ou du document filtre ASCII à utiliser (mode ASCII) ou * pour restaurer le jeu/filtre ASCII par défaut
typeFiltre	Numérique	→	0 = Filtre d'exportation, 1 = Filtre d'importation

Description

La commande UTILISER FILTRE permet de modifier le jeu de caractères utilisé par 4D pour toutes les opérations de transfert entre la base et un document ou un port série. Cela inclut les données transférées par les commandes d'import/export ASCII, SYLK et DIF, ainsi que celles envoyées par les commandes ENVOYER PAQUET et RECEVOIR PAQUET (paquets de type texte) et RECEVOIR BUFFER. Les filtres n'ont pas d'effet sur les données transférées par les commandes ENVOYER ENREGISTREMENT, ENVOYER VARIABLE, RECEVOIR ENREGISTREMENT, ENVOYER PAQUET et RECEVOIR PAQUET (paquets de type BLOB) et RECEVOIR VARIABLE.

La commande UTILISER FILTRE s'utilise différemment suivant que la base fonctionne en mode Unicode ou en mode compatibilité ASCII. Elle charge en mémoire un jeu de caractères ou un filtre ASCII.

Note : Pour plus d'informations sur ces modes, reportez-vous à la section Codes ASCII.

Mode Unicode
En mode Unicode, le paramètre filtre doit correspondre au nom “IANA” du jeu de caractères à utiliser, ou l’un de ses alias. Par exemple, les noms “iso-8859-1” ou “utf-8” sont des noms valides, ainsi que les alias “latin1” ou “l1”. Pour plus d’informations sur ces noms, reportez-vous à l’adresse http://www.iana.org/assignments/character-sets. Des exemples de noms IANA sont également fournis dans la description de la commande CONVERTIR DEPUIS TEXTE.

Mode compatibilité ASCII
Dans ce mode, la commande charge en mémoire le document de filtre ASCII filtre préalablement sauvegardé et l'utilise. Le filtre ASCII doit avoir été préalablement créé à l'aide d'une version antérieure de 4D. Si vous passez une chaîne vide dans le paramètre filtre, UTILISER FILTRE affiche une boîte de dialogue standard d'ouverture de fichiers pour que l'utilisateur puisse sélectionner un filtre ASCII existant.

Si typeFiltre est égal à 0, le filtre est défini pour l'exportation. Si typeFiltre est égal à 1, il est défini pour l'importation. Si vous ne passez pas le paramètre typeFiltre, le filtre d'exportation est utilisé par défaut.

Lorsque le paramètre * est passé, le jeu de caractères par défaut est rétabli (filtre d'importation ou d'exportation, en fonction de la valeur de typeFiltre).
En mode Unicode dans 4D v11, le jeu de caractères par défaut est UTF-8.
En mode compatibilité, l'ASCII Mac standard est rétabli.

Exemple

L'exemple suivant (mode Unicode) utilise le jeu de caractères UTF-16 pour exporter un texte, puis le jeu de caractères par défaut est rétabli :

```
UTILISER FILTRE ("UTF-16LE"; 0)  ` Utiliser le jeu de caractères UTF-16 'Little Endian'
EXPORTER TEXTE ([Ma Table]; "Mon Texte")  ` Exporter les données avec le filtre
UTILISER FILTRE (*; 0)  ` Rétablir le jeu par défaut
```

Référence

ECRITURE DIF, ECRITURE SYLK, ENVOYER PAQUET, EXPORTER TEXTE, IMPORTER TEXTE, LECTURE DIF, LECTURE SYLK, Mac vers Windows, RECEVOIR BUFFER, RECEVOIR PAQUET, Windows vers Mac.

Variables et ensembles système

La variable système OK prend la valeur 1 si le filtre est correctement chargé, sinon elle prend la valeur 0.

7

Compilateur

Le compilateur intégré de 4D vous permet de traduire vos applications de base de données en instructions de niveau assembleur. Les avantages procurés par le compilateur sont les suivants :

• **Vitesse :** votre base de données s'exécute de 3 à 1000 fois plus vite.

• **Vérification du code :** la cohérence interne du code de votre application de base de données est entièrement contrôlée. Les conflits de logique et de syntaxe sont détectés.

• **Protection :** une fois votre base compilée, vous pouvez en supprimer le code interprété. Alors, la base compilée dispose des mêmes fonctionnalités que la base originale, à la différence près que la structure et les méthodes ne peuvent plus être visualisées ni modifiées délibérément ou par inadvertance.

• **Application indépendantes "double-cliquables" :** une base compilée peut également être transformée en application indépendante (sous Windows, des fichiers ".EXE") comportant sa propre icône.

Pour une description du fonctionnement du compilateur de 4D, reportez-vous au manuel *Mode Développement*.

Les commandes de ce thème sont liées à l'utilisation du compilateur. Elles vous permettent de normaliser les types de données exploitées dans votre base. La commande APPELER 4D est utilisée spécifiquement dans les bases compilées.

C_BLOB	C_ENTIER	C_REEL	APPELER 4D
C_BOOLEEN	C_ENTIER LONG	C_ALPHA	
C_DATE	C_IMAGE	C_TEXTE	
C_GRAPHE	C_POINTEUR	C_HEURE	

A l'exception d'APPELER 4D, ces commandes déclarent des variables et leur assignent un type. La déclaration des variables permet de lever toute ambiguïté en ce qui concerne leur type. Lorsqu'une variable n'est pas déclarée par l'une de ces commandes, le compilateur déduit son type. Mais il lui est souvent difficile de déduire le type d'une variable utilisée dans les formulaires. Par conséquent, il est particulièrement important d'utiliser ces commandes pour déclarer les variables placées dans les formulaires.

Note : Pour gagner du temps, vous pouvez utiliser l'option de génération et de mise à jour des méthodes de typage (appelées “Méthodes compilateur”), proposée dans la fenêtre du compilateur. Cette option crée automatiquement des méthodes de typage recensant et donnant un type à l'ensemble des variables utilisées dans la base.

Les **tableaux** sont des variables devant respecter les mêmes règles que les variables standard en vue de la compilation. Les commandes de déclaration des tableaux sont groupées dans le thème “Tableaux”.

Principes généraux d'écriture de code destiné à être compilé

• Vous ne devez pas donner le même nom à des méthodes ou des variables différentes. Vous ne devez pas avoir une méthode qui aurait le même nom qu’une variable.

• Vous ne pouvez pas modifier le type d'une variable ou d'un tableau.

• Vous ne pouvez pas convertir un tableau simple en tableau à deux dimensions, et vice-versa.

• Vous ne pouvez pas modifier la longueur d'une variable chaîne ni celle des éléments d'un tableau alphanumérique.

• Bien que le compilateur déduise le type des variables si nécessaire, il est conseillé de déclarer le type des variables à l'aide des directives de compilation lorsque le type de données est ambigu, en particulier dans un formulaire.

• Une autre raison de déclarer explicitement le type des variables est l'optimisation de votre code. Cela est particulièrement vrai pour les variables utilisées comme compteurs. Dans ce cas, l'utilisation de variables de type Entier long assure un maximum de performances.

• Pour effacer une variable (c'est-à-dire l'initialiser à une valeur nulle), utilisez la commande EFFACER VARIABLE avec le nom de la variable. N'utilisez pas de chaîne alphanumérique pour désigner le nom de la variable avec la commande EFFACER VARIABLE.

• La fonction Indefinie retournera toujours Faux. Les variables sont toujours définies.

• Les opérations numériques effectuées sur des variables de type Entier long ou Entier sont généralement beaucoup plus rapides que celles effectuées sur des valeurs Numérique (réel).

• Les indirections de variables, utilisées dans la version 3 de 4D, ne sont pas permises. Vous ne pouvez pas utiliser l'indirection alphanumérique, à l'aide du symbole 'paragraphe' (§), pour référencer des variables indirectement. Vous ne pouvez pas non plus utiliser les indirections numériques, à l'aide des accolades ({...}). Les accolades ne peuvent être utilisées que pour accéder à un élément de tableau ayant été déclaré. En revanche, vous pouvez utiliser le passage de paramètres.

Ces principes sont détaillés dans les sections suivantes :
• Utilisation des directives de compilation, expliquant quand et où écrire des directives de compilation
• Guide du typage, décrivant les différents types de conflits pouvant se produire lors de la compilation des bases 4D,

• Précisions de syntaxe, fournissant des informations supplémentaires concernant plusieurs commandes 4D,
• Conseils d'optimisation, proposant des conseils permettant d'accélérer l'exécution des applications en mode compilé.

Exemples

(1) Voici quelques déclarations de variables standard pour le compilateur :

```
C_BLOB(vxMonBlob)  ` La variable process vxMonBlob est déclarée avec le type BLOB
C_BOOLEEN(<>SousWindows)  ` La variable interprocess <>SousWindows est déclarée avec le type booléen
C_DATE($vdCurDate)  ` La variable locale $vdCurDate est déclarée avec le type Date
C_GRAPHE(vg1;vg2;vg3)  ` Les 3 variables process vg1, vg2 et vg3 sont déclarées avec le type Graphe
```

(2) Dans cet exemple, la méthode projet uneMéthodeParmiD'Autres déclare 3 paramètres:

```
` Méthode projet uneMéthodeParmiD'Autres
` uneMéthodeParmiD'Autres ( Numérique ; Entier { ; Entier long } )
` uneMéthodeParmiD'Autres ( Montant ; Pourcentage { ; Ratio } )

C_REEL($1)  ` le 1er paramètre est du type Réel (Numérique)
C_ENTIER($2)  ` le 2e paramètre est du type Entier
C_ENTIER LONG($3)  ` le 3e paramètre est du type Entier long

` ...
```

(3) Dans l'exemple suivant, la méthode projet ajoutCapitale accepte un paramètre de type Chaîne et retourne une chaîne :

```
` Méthode projet ajoutCapitale
` ajoutCapitale ( Alpha ) -> Alpha
` ajoutCapitale ( Chaîne source ) -> Chaîne avec la première lettre capitale

C_ALPHA(255;$0;$1)
$0:=Majusc(Sous chaine($1;1;1))+Minusc(Sous chaine($1;2))
```

(4) Dans l'exemple suivant, la méthode projet envoyerPaquets accepte un paramètre de type Heure suivi d'un nombre variable de paramètres de type Texte :

```
` Méthode projet envoyerPaquets
` envoyerPaquets ( Heure ; Texte { ; Texte2... ; TextN } )
` envoyerPaquets ( docRef ; Données { ; Données2... ; DonnéesN } )

C_HEURE ($1)
C_TEXTE (${2})
C_ENTIER LONG ($vlPaquet)

Boucle ($vlPaquet;2;Nombre de parametres)
   ENVOYER PAQUET ($1;${$vlPaquet})
Fin de boucle
```

(5) Dans l'exemple suivant, la méthode projet compiler_Param_Prédéclare28 pré-déclare la syntaxe d'autres méthodes projet, à l'intention du compilateur :

```
` Méthode projet compiler_Param_Prédéclare28

C_REEL(uneMéthodeParmiDautres;$1)  ` uneMéthodeParmiDautres ( Réel ; Entier { ; Entier long } )
C_ENTIER(uneMéthodeParmiDautres;$2)  ` ...
C_ENTIER LONG(uneMéthodeParmiDautres;$3)  ` ...
C_ALPHA(ajoutCapitale;255;$0;$1)  ` ajoutCapitale ( Alpha ) -> Alpha
C_HEURE(envoyerPaquets;$1)  ` envoyerPaquets ( Heure ; Texte { ; Texte2... ; TexteN } )
C_TEXTE(envoyerPaquets;${2})  ` ...
```

Référence

APPELER 4D, C_ALPHA, C_BLOB, C_BOOLEEN, C_DATE, C_ENTIER, C_ENTIER LONG, C_GRAPHE, C_HEURE, C_IMAGE, C_POINTEUR, C_REEL, C_TEXTE.

Typage des variables

4D utilise trois catégories de variables :
• les variables locales,
• les variables process,
• les variables interprocess.

Pour plus d'informations sur ce point, reportez-vous à la section Variables. Les variables process et les variables interprocess sont structurellement de même nature pour le compilateur.

Types des variables
Toutes les variables ont un type. Comme décrit dans la section Types de données, vous disposez de 12 types pour les variables simples :
Booléen
Alphanumérique (ou Chaîne fixe)
Date
Entier
Entier long
Graphe
Heure
Image
Numérique (ou Réel)
Pointeur
Texte
BLOB

Pour les variables de type Tableau, vous disposez des 9 types suivants :
Tableau Booléen
Tableau Alpha (ou Chaîne fixe)
Tableau Date
Tableau Entier
Tableau Entier long
Tableau Image
Tableau Numérique (ou Réel)
Tableau Pointeur
Tableau Texte

Création de la table des symboles
Lorsque vous travaillez avec 4D en mode interprété, une variable peut avoir plusieurs types. Cette tolérance se justifie parfaitement puisque vous êtes en mode interprété. En effet, à chaque ligne de code, 4D interprète l'instruction et comprend le contexte.
Lorsque vous travaillez en mode compilé, vous êtes dans une situation différente. Alors que l'interprétation agit ligne par ligne, la compilation s'intéresse à une base dans sa globalité.

La manière d'opérer du compilateur est la suivante :
• Le programme explore systématiquement les objets qui lui sont proposés par 4D. Ces objets sont les méthodes base, les méthodes projet, les méthodes formulaire, les méthodes table (triggers) et les méthodes objet.
• Le programme peigne ces objets pour retrouver le type de chacune des variables utilisées dans la base et génère la table des variables et des méthodes.
• Une fois qu'il a retrouvé le type de toutes les variables, le compilateur traduit la base. Encore faut-il qu'il puisse identifier un type d'une manière univoque pour chacune des variables.

Si le compilateur trouve un même nom de variable avec deux types différents, il n'a aucune raison de privilégier l'un par rapport à l'autre. En d'autres termes, avant de pouvoir ranger un objet, de lui donner une adresse mémoire, le compilateur a besoin de connaître l'identité exacte de cet objet, c'est-à-dire son nom et son type, le type permettant au compilateur de déduire sa taille. Ainsi, pour chaque application compilée, le compilateur crée un plan, contenant, pour chaque variable, son nom (ou identificateur), son emplacement (ou adresse mémoire) et la taille qu'elle occupe (représentée par son type). Ce "plan" s'appelle la table de symboles. Une option des Préférences vous permet de générer ou non cette table sous forme de fichier lors de la compilation.
Ce plan est également utilisé pour la génération automatique des méthodes compilateur.

Typage des variables
Le compilateur doit respecter les critères d'identification des variables.
Il existe deux possibilités :
• Si les variables ne sont pas typées, le compilateur s'en occupera automatiquement pour vous. Toutes les fois que c'est possible, dans la mesure où il n'y a pas d'ambiguïtés, le compilateur déduit automatiquement pour vous le type des variables utilisées. Si, par exemple, vous écrivez :

```
V1 := Vrai
```

le compilateur pourra en déduire que la variable V1 est de type Booléen.

De même, si vous écrivez :

```
V2:= "Ceci est une phrase exemple"
```

le compilateur en déduira que V2 est une variable de type Texte.

Le programme peut également déduire le type des variables dans des cas moins faciles :

```
V3:= V1`V3 est du même type que V1
V4:= 2*V2          `V4 est du même type que V2
```

Le compilateur déduit également le type de vos variables d'après les appels aux commandes 4D et à vos propres méthodes. Si vous passez à une méthode un paramètre de type Booléen et un paramètre de type Date, le compilateur donnera le type Booléen et le type Date aux variables locales $1 et $2 de la méthode appelée.

Lors de ces déductions, sauf indication contraire dans les Préférences, le compilateur ne donne pas à vos variables des types limitatifs : Entier, Entier long ou Alphanumérique. Par défaut, le compilateur donne toujours le type le plus large possible à vos variables. Si vous écrivez :

```
LeNombre :=4
```

le compilateur donnera à la variable *LeNombre* le type Réel, même si en toute rigueur la valeur 4 est entière. En d'autres termes, le compilateur n'exclut pas que dans d'autres occurrences de la variable, la valeur puisse être 4,5.

Si vous ne voulez pas de cette interprétation générale, vous pouvez préciser vos choix : c'est l'objet de ce qu'on appelle les *directives de compilation*.

• Les directives de compilation servent à déclarer de façon explicite les variables simples que vous utilisez dans vos bases.
Leur utilisation se fait de la façon suivante :

C_BOOLEEN(Var)

Par cette directive, vous forcez le compilateur à créer une variable *Var* dont le type sera Booléen.
Dans le cas où une application comporte des directives de compilation, le compilateur les détecte et n'a pas à se livrer à une quelconque estimation.

Une directive a la priorité sur une déduction à partir d'une affectation ou d'une utilisation.

Les variables simples déclarées par la directive de compilation C_ENTIER sont considérés comme des Entiers longs, compris entre –2147483648 et +2147483647 comme pour les variables déclarées par la directive C_ENTIER LONG.

Quand utiliser des directives de compilation ?

Les directives de compilation sont utiles dans deux cas :
• lorsque le compilateur ne peut pas déduire seul le type d'une variable,
• lorsque vous voulez éviter que le compilateur fasse des déductions.
Par ailleurs, utiliser des directives de compilation peut vous permettre de réduire le temps de compilation.

Cas d'ambiguïté

Il arrive que le compilateur ne puisse pas déduire le type d'une variable, et cela pour plusieurs raisons. Il est impossible de recenser tous les cas de figure. Une chose est certaine, c'est qu'en cas d'impossibilité à compiler, le compilateur vous en donnera la raison précise ainsi que les moyens d'y remédier.

On peut cependant distinguer trois causes majeures d'hésitation pour le compilateur : l'ambiguïté proprement dite, l'ambiguïté sur une déduction forcée, et l'impossibilité totale de déduire un type.

• *L'ambiguïté proprement dite*
L'ambiguïté sur le nom de la variable est généré dans le cas suivant : le compilateur choisit la première variable qu'il rencontre et assigne arbitrairement à la suivante, de même nom mais de type différent, le type qu'il a précédemment attribué.

Prenons un exemple simple :

dans une méthode A,

```
LaVariable:=Vrai
```

dans une méthode B,

```
LaVariable:="La lune est verte"
```

Dans le cas où la méthode A est compilée avant la méthode B, le compilateur considérera que LaVariable:="La lune est verte" est un changement de type d'une variable précédemment rencontrée. Il vous signalera qu'il y a retypage. Il génère une erreur qu'il vous appartient de corriger.
Ne vous inquiétez pas, le compilateur ne transformera pas votre variable LaVariable:="La lune est verte" en variable booléenne sans vous demander ce que vous en pensez !

• *L'ambiguïté sur une déduction forcée*
Il peut arriver que le compilateur déduise un type sur un objet qui ne convient pas à son utilisation finale. Dans ce cas, vous devrez typer explicitement vos variables à l'aide des directives de compilation.

Voici un exemple de cas d'ambiguïté lors de l'utilisation des listes de valeurs par défaut sur un objet : dans les formulaires, il est possible de spécifier une liste de valeurs par défaut pour les objets de type combo box, pop up menu, menu/liste déroulante, onglet, zone de défilement et liste déroulante à l'aide du bouton d'édition **Valeurs** (voir à ce sujet le manuel *Mode Développement* de 4D). Les valeurs par défaut sont automatiquement chargées dans un tableau dont le nom est le même que celui de l'objet.
Dans le cas simple où l'objet n'est pas utilisé dans une méthode, le compilateur peut déduire son type sans ambiguïté et l'objet sera typé en tableau texte par défaut.
En revanche, dans le cas où vous devez initialiser la position de votre tableau pour son affichage dans le formulaire, vous pouvez écrire les instructions suivantes dans la méthode formulaire :

```
Au cas ou
   : (Evenement formulaire=Sur chargement)
      MonPopUp:=2
      ...
Fin de cas
```

C'est dans ce cas que l'ambiguïté apparaît : lors de l'analyse des méthodes, le compilateur déduira par défaut le type Réel pour la variable *MonPopUp*. Dans ce cas très précis, vous devez explicitement déclarer le tableau dans une méthode compilateur ou dans la méthode formulaire :

```
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU TEXTE(MonPopUp;2)
        MonPopUp:=2
        ...
Fin de cas
```

• *L'impossibilité totale de déduire un type*
Dans ce cas, seule une directive de compilation peut orienter le compilateur. Le compilateur peut se trouver dans cette situation lorsqu'une variable est utilisée sans être déclarée et dans un cadre qui ne donne aucune information sur son type possible.
Le phénomène se produit principalement dans quatre cas :
- lorsque vous utilisez des pointeurs,
- lorsque vous utilisez une commande à syntaxe multiple,
- lorsque vous utilisez une commande 4D avec des paramètres optionnels de types différents,
- lorsque vous utilisez une méthode appelée via un URL.

- Cas des pointeurs
Un pointeur étant un outil universel qui a donc pour première caractéristique la flexibilité, il est inutile d'espérer qu'il renvoie un type d'une manière ou d'une autre, hormis le sien propre.
Supposons que vous écriviez dans une méthode la séquence suivante :

```
LaVar1:=5,2          (1)
LePointeur:=->LaVar1 (2)
LaVar2:=LePointeur-> (3)
```

Bien que la ligne (2) définisse le type de la variable pointée par le pointeur *LePointeur*, *LaVar2* n'est pas pour autant typée. Lors de la compilation, le compilateur peut reconnaître un pointeur, mais n'a aucun moyen de savoir sur quel type de variable il pointe. Il ne peut donc pas déduire le type de *LaVar2* ; une directive de compilation du type C_REEL(LaVar2) est donc indispensable.

- Cas des commandes à syntaxe multiple
Lorsque vous utilisez une variable associée à la fonction Annee de, la variable ne peut être, compte tenu de la nature même de la fonction, que de type Date. En revanche, prenons un cas extrême : la commande LIRE PROPRIETES CHAMP admet deux syntaxes :
LIRE PROPRIETES CHAMP(NoTable;NoChamp;Type;Longueur;Indexée)
LIRE PROPRIETES CHAMP(Pointeur_Champ;Type;Longueur;Indexée)

Lorsque vous utilisez cette commande, le compilateur ne peut pas deviner quelle syntaxe et quels paramètres vous avez choisis. Il vous appartient alors d'orienter le compilateur par une directive de compilation.

- Cas des commandes 4D ayant des paramètres optionnels de types différents
Lorsque vous utilisez une commande 4D qui accepte plusieurs paramètres optionnels de différents types, le compilateur ne peut pas deviner quels paramètres ont été passés.
Par exemple, la commande INFORMATION ELEMENT admet deux paramètres optionnels. Le premier est de type Entier long, le second est de type Booléen.
La commande peut donc être utilisée comme ceci :
INFORMATION ELEMENT(liste;position;num;texte;sous-liste;déployé)
ou comme cela :
INFORMATION ELEMENT(liste;position;num;texte;déployé)
Vous devez donc utiliser des directives de compilation pour typer les paramètres optionnels passés à la commande (s'ils n'ont pas déjà été typés suite à leur utilisation dans un autre endroit de la base).

- Cas des méthodes appelées via un URL
Si vous écrivez des méthodes appelées via un URL, il est nécessaire de déclarer explicitement la variable Texte $1 dans vos méthodes, par l'intermédiaire de l'instruction C_TEXTE($1), dans le cas où vous n'utilisez pas $1 dans la méthode. En effet, le compilateur ne peut pas deviner qu'une méthode 4D va être appelée via un URL.

Réduction du temps de compilation
Si toutes les variables utilisées dans votre base sont explicitement déclarées, il n'est pas nécessaire que le compilateur refasse tout le typage. Dans ce cas, vous pouvez lui demander d'effectuer uniquement la phase de traduction de vos méthodes dans le chemin de compilation. Ainsi, vous économiserez environ 50 % du temps de compilation.

Cas d'optimisation
Les directives de compilation peuvent vous aider à accélérer vos méthodes. Pour plus de précision à ce sujet, reportez-vous à la section Conseils d'optimisation. Pour nous en tenir à un exemple simple dans cette section de présentation, imaginez que vous incrémentiez un compteur. Si vous n'avez pas déclaré la variable, le compilateur considérera par défaut qu'elle est de type Numérique (ou réel). Si vous prenez soin de préciser qu'il s'agit d'un Entier, l'exécution de la base compilée sera plus satisfaisante. En effet, un Réel occupe, sur PC par exemple, 8 octets en mémoire alors que si vous choisissez un type limitatif, Entier ou Entier long, le compteur n'en occupera que 4. Il est bien évident que l'incrémentation d'un compteur de 8 octets est plus longue que celle d'un compteur de 4 octets.

Où placer les directives de compilation ?

Vous avez deux possibilités selon que vous voulez que le compilateur vérifie ou non votre typage.

Typage des variables par le compilateur
Si vous voulez que le compilateur vérifie votre typage ou bien s'en charge lui-même, placer une directive de compilation est simple. Vous avez le choix entre deux possibilités, qui correspondent d'ailleurs à deux méthodes de travail :
• ou bien vous écrivez cette directive lors de la première utilisation de la variable, qu'il s'agisse d'une variable locale, process ou interprocess. La seule recommandation en la matière est que vous l'inscriviez bien à la première utilisation de la variable dans la première méthode exécutée. Attention, lors de la compilation le compilateur prend les méthodes dans l'ordre de leur création, et non dans l'ordre dans lequel elles apparaissent dans l'Explorateur.
• ou bien, si vous êtes systématique, regroupez toutes vos variables process ou interprocess avec les directives afférentes dans la Méthode base Sur ouverture ou une méthode appelée par la Méthode base Sur ouverture.
Pour les variables locales, regroupez ces directives en tête de la méthode où elles sont utilisées.

Typage assuré par vos soins
Si vous voulez que le compilateur n'ait pas à vérifier votre typage, vous devez lui donner les clés de l'identification de ses objets.
La convention à respecter est la suivante : les directives de compilation des variables process ou interprocess, ainsi que les paramètres, devront être placées dans une ou plusieurs méthodes dont le nom commence par **Compiler**.
Par défaut, le compilateur vous permet de générer automatiquement cinq types de méthodes Compilateur regroupant les directives pour les variables, les tableaux et les paramètres des méthodes (pour plus d'informations sur ce point, reportez-vous au manuel *Mode Développement*).

Note : La déclaration des paramètres des méthodes obéit à la syntaxe suivante : Directive (nom de méthode; param). Cette syntaxe n'est pas exécutable en mode interprété.

Paramètres particuliers
• Les paramètres reçus par les méthodes base
Ces paramètres sont typés par défaut par le compilateur, si la déclaration n'a pas été faite explicitement. Néanmoins, si vous les déclarez, la déclaration doit se faire à l'intérieur des méthodes base.
La déclaration de ces paramètres ne peut pas se faire dans une méthode compilateur.
Exemple : la Méthode base Sur connexion Web reçoit six paramètres $1 à $6 de type texte. En début de méthode, vous devez écrivez : C_TEXTE($1;$2;$3;$4;$5;$6)

• Les triggers
Le paramètre $0 (Entier long), résultat d'un trigger, est typé par défaut par le compilateur, si la déclaration n'a pas été faite explicitement. Néanmoins si vous le déclarez, la déclaration doit se faire à l'intérieur du trigger.
La déclaration de ce paramètre ne peut pas se faire dans une méthode compilateur.

• Les objets acceptant l'événement formulaire "Sur glisser"
Le paramètre $0 (Entier long), résultat d'un événement formulaire "Sur glisser", est typé par défaut par le compilateur, si la déclaration n'a pas été faite explicitement. Néanmoins si vous le déclarez, la déclaration doit se faire à l'intérieur de la méthode objet.
La déclaration de ce paramètre ne peut pas se faire dans une méthode compilateur.

Note : Le compilateur n'initialise pas le paramètre $0. Dès que vous utilisez l'événement formulaire Sur glisser, vous devez donc initialiser $0. Par exemple :

```
C_ENTIER LONG($0)
Si (Evenement formulaire=Sur glisser)
    $0:=0
    ...
    Si ($TypeDeDonnées=Est une image)
        $0:=-1
    Fin de si
    ...
Fin de si
```

Une syntaxe particulière : C_ALPHA
La syntaxe de toutes les directives de compilation est extrêmement simple ; seule la commande C_ALPHA requiert une attention particulière puisqu'elle admet un paramètre supplémentaire : la longueur maximale de la chaîne.
C_ALPHA(Longueur;Var1{;Var2;…;VarN})

Par définition, C_ALPHA porte sur des chaînes fixes, il est donc naturel de donner la longueur de cette chaîne. A cet égard, il convient de rappeler une différence de comportement entre une base interprétée et une base compilée.
Dans une base interprétée, la séquence suivante :

```
LaLongueur:=15
C_ALPHA(LaLongueur; LaChaine)
```

serait parfaitement logique. 4D interpréterait *LaLongueur* puis remplacerait *LaLongueur* par sa valeur dans la directive.
En revanche, le compilateur utilise cette commande durant le typage des variables et en dehors de toute affectation particulière. Il ne peut donc pas savoir que *LaLongueur* est égale à 15. Ne connaissant pas la longueur de la chaîne, il ne peut pas lui réserver de place dans la table des symboles. Par conséquent, dans l'optique d'une compilation, il convient d'utiliser une constante pour spécifier la longueur de la chaîne de caractères déclarée. Cette déclaration se fera donc de la façon suivante :

```
C_ALPHA(15;LaChaine)
```

Il en va de même pour la déclaration de tableaux d'alphas par la commande :

```
TABLEAU ALPHA(LaLongueur;LeTableau;Nombre d'éléments)
```

Le paramètre indiquant la longueur des chaînes du tableau sera une constante.

En revanche, vous pouvez utiliser les constantes 4D ou des valeurs hexadécimales pour spécifier la longueur des chaînes dans ces deux directives de compilation. Exemple :

```
C_ALPHA(Constante4D;LaChaîne)
TABLEAU ALPHA(Constante4D;LeTableau;2)
C_ALPHA(0x000A;LaChaîne)
TABLEAU ALPHA(0x000A;LeTableau;2)
```

Ne confondez pas la longueur d'un champ Alphanumérique qui peut être au maximum de 80 caractères avec une variable Alphanumérique. La longueur d'une chaîne déclarée par la directive C_ALPHA ou appartenant à un TABLEAU ALPHA peut être comprise entre 1 et 255.

Note : La syntaxe de cette commande vous permet de déclarer plusieurs variables de même longueur en une seule ligne. Si vous souhaitez déclarer plusieurs chaînes fixes de longueur différente, il faudra le faire en plusieurs lignes.

Une liberté permise par le compilateur

Les directives de compilation lèvent toute ambiguïté sur les types et, pour le cas des chaînes alphanumériques, sur les longueurs. L'exigence de rigueur ne se fait pas pour autant intolérance.

S'il vous arrive d'utiliser un Numérique là où vous avez déclaré un Entier ou de manipuler une chaîne de 30 caractères là où vous en avez déclaré une de 10, le compilateur ne considère pas qu'il y a conflit et suit simplement vos directives. Ainsi, si vous écrivez :

```
C_ENTIER(vEntier)
vEntier:=2,6
```

Le compilateur ne verra pas un conflit de types de nature à empêcher la compilation et prendra automatiquement en compte la partie entière arrondie du nombre affecté (3 au lieu de 2,6).

De la même façon, s'il vous arrive de déclarer une chaîne plus courte que la chaîne que vous manipulez, le compilateur ne prend que le nombre de caractères déclarés. Ainsi, dans la séquence suivante :

```
C_ALPHA(10;LaChaine)
LaChaine:="Il fait très beau aujourd'hui"
```

le compilateur prend en compte les 10 premiers caractères de la constante, soit "Il fait tr".

Référence

Conseils d'optimisation, Guide du typage, Messages d'erreurs, Précisions de syntaxe.

Cette section décrit les principales causes de conflits de types sur les variables, ainsi que les manières de les éviter.

Conflits sur les variables simples

Les conflits de types simples peuvent se résumer comme suit :
• conflit entre deux utilisations,
• conflit entre une utilisation et une directive de compilation,
• conflit par retypage implicite,
• conflit entre deux directives de compilation.

Conflit entre deux utilisations
Le conflit de types le plus simple est celui pour lequel un même nom de variable désigne deux objets différents. Imaginez que dans une application, vous écriviez :

```
LaVariable:=5
```

et que, quelque part ailleurs, dans la même application, vous écriviez :

```
LaVariable:=Vrai
```

Vous générez un conflit de types. Le remède est simple : renommez l'une des deux variables.

Conflit entre une utilisation et une directive de compilation
Imaginez que dans une application, vous écriviez :

```
LaVariable:=5
```

et que, quelque part ailleurs, dans la même application, vous écriviez :

```
C_BOOLEEN(LaVariable)
```

Le compilateur, peignant d'abord les directives de compilation, fera de *LaVariable* un Booléen mais lorsqu'il découvrira :

```
LaVariable:=5
```

il signalera un conflit de types. Ici encore, le remède est simple : renommez votre variable ou modifiez la directive de compilation.

L'utilisation de variables de types différents dans une expression génère des incohérences. Le compilateur signale très logiquement les incompatibilités. Prenons un exemple simple. Vous écrivez :

```
vBooléen:=Vrai   `Le compilateur déduit que vBooléen est de type Booléen
C_ENTIER(<>vEntier)   `Déclaration d'un Entier par une directive de compilation
<>vEntier:=3   `Commande compatible avec la directive de compilation
LaVar:= <>vEntier+vBooléen
    `Opération contenant des variables dont les types sont incompatibles
```

Conflit par retypage implicite
Certaines fonctions renvoient des variables d'un type bien précis. L'affectation du résultat d'une de ces variables à une variable déjà typée différemment provoquera un conflit de types si vous ne faites pas attention.
Dans une application interprétée, vous pouvez écrire :

```
No_Ident:=Demander("Numéro d'identification")   `No_Ident est de type Texte
Si(Ok=1)
    No_Ident:=Num(No_Ident)   `No_Ident est de type Numérique
    CHERCHER([Personnes]Id= No_Ident)
Fin de si
```

Vous générez ici un conflit de type sur la troisième ligne. Le remède consiste à contrôler le comportement de la variable. Dans certains cas, vous aurez à créer des variables intermédiaires d'un nom différent. Dans d'autres cas, comme celui-ci en particulier, vous pouvez structurer différemment votre méthode :

```
No_Ident:=Num(Demander("Numéro d'identification"))
    `No_Ident est de type Numérique
Si(Ok=1)
    CHERCHER([Personnes]Id=No_Ident)
Fin de si
```

Conflit entre deux directives de compilation
Déclarer deux fois la même variable par deux directives de compilation différentes constitue, bien sûr, un retypage. Si, dans la même base, vous écrivez :

```
C_BOOLEEN(LaVariable)
C_TEXTE(LaVariable)
```

le compilateur est confronté à un dilemme et vous demande quelles étaient vos intentions. Le remède est simple : renommez l'une des deux variables.

Il ne faut pas oublier que le conflit de types pour une directive C_ALPHA peut surgir si vous modifiez la longueur maximale de la chaîne de caractères. Ainsi, si vous écrivez :

```
C_ALPHA(5;MaChaine)
MaChaine:="Fleur"
C_ALPHA(7;MaChaine)
MaChaine:="Bonjour"
```

le compilateur est dans une situation de conflit puisque dans la déclaration des variables de type Alphanumérique, il doit réserver un emplacement de taille adéquate.
Dans ce cas, le remède consiste à donner une directive de compilation qui prenne la longueur maximale, puisque par défaut, le compilateur acceptera une longueur inférieure. Vous pouvez donc écrire :

```
C_ALPHA(7;LaChaine)
LaChaine:="Fleur"
LaChaine:="Bonjour"
```

Note : Si vous aviez écrit deux fois C_ALPHA(7;LaChaine), c'est-à-dire :

```
C_ALPHA(7;LaChaine)
LaChaine:="Fleur"
C_ALPHA(7;LaChaine)
LaChaine:="Bonjour"
```

le compilateur l'accepterait tout à fait. C'est simplement redondant.

Note sur les variables locales

Les conflits de types pour les variables locales sont absolument identiques aux conflits de types pour les variables process et interprocess, à ceci près que ces conflits se déroulent dans un espace plus restreint.

Les conflits se jouent au niveau général de la base pour les variables process et interprocess. Les conflits se jouent au niveau particulier de la méthode pour les variables locales. Vous ne pouvez donc pas écrire dans la même méthode :

```
$Temp:="Bonjour"
```

et puis plus loin

```
$Temp:=5
```

En revanche, vous pouvez écrire dans une méthode M1 :

```
$Temp:="Bonjour"
```

et dans une méthode M2 :

```
$Temp:=5
```

Conflits sur les variables tableau

Les conflits possibles pour un tableau ne portent jamais sur la taille du tableau. En mode compilé comme en mode interprété, les tableaux sont gérés dynamiquement. La taille d'un tableau peut varier au fil des méthodes et vous n'avez pas, bien sûr, à déclarer une taille maximale pour un tableau.

Vous pouvez, en conséquence, dimensionner un tableau à zéro, ajouter ou retirer des éléments, en effacer le contenu. Dans l'optique de la compilation, et ce, dans une même méthode, pour un tableau local ou dans toute la base pour un tableau process ou interprocess, vous devez veiller aux points suivants :

- ne pas changer le type des éléments du tableau,
- ne pas changer le nombre de dimensions d'un tableau,
- dans le cas des tableaux Alpha, ne pas changer la longueur des chaînes de caractères.

Changement de type des éléments d'un tableau

Si vous déclarez un tableau comme étant un tableau d'Entiers, il doit rester un tableau d'Entiers pour toute la base. Il ne pourra jamais contenir, par exemple, des éléments de type Booléen.

Si, dans une application, vous écrivez :

```
TABLEAU ENTIER(LeTableau;5)
TABLEAU BOOLEEN(LeTableau;5)
```

le compilateur ne peut identifier pour vous le type de *LeTableau*.

Renommez simplement l'un des deux tableaux.

Changement du nombre de dimensions d'un tableau
En version interprétée, il peut vous arriver de changer le nombre de dimensions d'un tableau.
Lorsque le compilateur établit sa table des symboles, il gère différemment les tableaux à une dimension et les tableaux à deux dimensions.
En conséquence, un tableau déclaré une fois comme étant un tableau à une dimension ne peut être re-déclaré ou utilisé comme un tableau à deux dimensions et vice versa.
Dans une même base, vous ne pouvez donc pas avoir :

TABLEAU ENTIER(LeTableau1;10)
TABLEAU ENTIER(LeTableau1;10;10)

En revanche, vous pouvez, évidemment, avoir dans la même application

TABLEAU ENTIER(LeTableau1;10)
TABLEAU ENTIER(LeTableau2;10;10)

Par ailleurs, vous pouvez parfaitement écrire :

TABLEAU BOOLEEN(LeTableau;5)
TABLEAU BOOLEEN(LeTableau;10)

Comme vous pouvez le noter dans nos exemples, c'est le nombre de dimensions d'un tableau qu'on ne peut changer en cours d'application et non la valeur des dimensions du tableau.

Note : Un tableau à deux dimensions est en fait un ensemble de plusieurs tableaux à une dimension. Pour plus de précisions, reportez-vous à la section Tableaux à deux dimensions.

Cas des tableaux de chaînes fixes
Les tableaux de chaînes fixes, aussi appelés tableaux Alpha, suivent la même règle que les variables Alphanumériques et pour les mêmes raisons.
Si vous écrivez :

TABLEAU ALPHA(5;LeTableau;10)
TABLEAU ALPHA(10;LeTableau;10)

le compilateur détecte un conflit de longueur. Le remède est simple : comme dans les chaînes Alpha, vous déclarez la longueur maximale. Le compilateur gère automatiquement les longueurs inférieures.

Retypages implicites
Lors de l'utilisation des commandes COPIER TABLEAU, ENUMERATION VERS TABLEAU, TABLEAU VERS ENUMERATION, SELECTION VERS TABLEAU, SELECTION LIMITEE VERS TABLEAU, TABLEAU VERS SELECTION, VALEURS DISTINCTES, vous pouvez, volontairement ou involontairement, être conduit à des changements de type d'éléments ou de nombre de dimensions, ou pour un tableau Alpha, à des changements de longueur de chaîne. Vous vous retrouverez donc dans un des trois cas cités précédemment.
Le compilateur vous délivrera un message d'erreur et la correction que vous aurez à faire sera en général évidente. Des exemples de retypage implicite de tableaux sont fournis dans la section Précisions de syntaxe.

Tableaux locaux
Si vous souhaitez compiler une base de données qui utilise des tableaux locaux (tableaux visibles uniquement par les méthodes qui les ont créés), il est nécessaire de les déclarer explicitement dans 4D avant de les utiliser.
La déclaration explicite d'un tableau signifie l'utilisation d'une commande de type TABLEAU REEL, TABLEAU ENTIER, etc.
Par exemple, si une méthode génère un tableau local d'entiers contenant 10 valeurs, vous devez écrire au préalable la ligne suivante :

```
TABLEAU ENTIER($MonTableau;10)
```

Typage des variables dessinées dans les formulaires

Les variables dessinées dans un formulaire, qu'il s'agisse d'une case à cocher ou d'une zone externe, sont toutes des variables soit process, soit interprocess.
En mode interprété, la question des types de ces variables ne se pose pas dans la pratique. Elle peut se poser, en revanche, dans l'optique d'une application compilée. Les règles du jeu sont cependant transparentes :
• soit vous avez typé votre variable,
• soit le compilateur lui attribue un type par défaut qui peut être défini dans les Préférences de compilation (voir le manuel *Mode Développement*).

Variables considérées par défaut comme des Numériques
Les variables suivantes sont considérées comme des Numériques par défaut :
Case à cocher
Case a cocher 3D
Bouton
Bouton inversé
Bouton invisible
Bouton 3D
Bouton image
Grille de boutons
Bouton radio
Bouton radio 3D
Radio image
Menu image
Menu déroulant hiérarchique
Liste hiérarchique
Règle
Cadran
Thermomètre

Note : Les variables de type Règle, Cadran et Thermomètre seront toujours typées comme des Numériques, même si vous avez choisi l'option Entier long par défaut pour le type des boutons dans les Préférences.

Pour ces variables, vous ne pouvez jamais vous trouver dans un conflit de types puisqu'elles ne peuvent avoir d'autres types que ce type-là, qu'on soit en interprété ou en compilé.

Les seuls conflits de types possibles sur l'une de ces variables viendraient du fait que le nom d'une variable serait le même que celui d'une autre variable placée à un autre endroit dans l'application. Dans ce cas, le remède consiste à renommer cette deuxième variable.

Variable Graphe
Une zone de graphe a automatiquement le type Graphe (Entier long). Il ne peut jamais y avoir de conflit de type.
Les seuls conflits de type possibles sur une variable de type Graphe viendraient du fait que le nom de cette variable serait le même que celui d'une autre variable placée à un autre endroit dans l'application. Dans ce cas, le remède consiste à renommer cette deuxième variable.

Variable Zone externe
Une zone externe est toujours un Entier long. Il ne peut jamais y avoir de conflit de types.
Les seuls conflits de types possibles sur une variable Zone externe viendraient du fait que le nom de cette variable serait le même que celui d'une autre variable placée à un autre endroit dans l'application. Dans ce cas, le remède consiste à renommer cette deuxième variable.

Variables considérées par défaut comme du Texte
Ces variables sont les suivantes :
Variable non-saisissable,
Variable saisissable,
Liste déroulante,
Menu/liste déroulante,
Zone de défilement,
Combo box,
Pop-up Menu,
Onglet.

Ces variables se divisent en deux catégories :
• les variables simples (variables saisissables et variables non-saisissables),
• les variables d'affichage de tableaux (listes déroulantes, menus/listes déroulantes, zone de défilement, pop-up menu, combo box, onglets).

• Variables simples
Par défaut, ces variables reçoivent le type Texte. Si elles sont utilisées dans une méthode objet ou une méthode formulaire, c'est le type que vous avez choisi qui leur sera attribué. Vous n'avez aucun risque de conflit de types autre que celui qui serait généré par une attribution préalable du même nom à une autre variable.

• Variables d'affichage de tableaux
De nombreuses variables vous servent à afficher des tableaux dans les formulaires. Si les valeurs par défaut ont été saisies au niveau des contrôles de saisie des variables en mode Développement, il est conseillé de typer les variables correspondantes explicitement par une déclaration de type tableau (TABLEAU ALPHA, TABLEAU TEXTE...).

Questions de type autour des pointeurs

Si vous utilisez des pointeurs dans vos applications, vous avez pu profiter de la puissance et de la flexibilité de cet outil dans 4D. Le compilateur en conserve intégralement les avantages.
Un pointeur peut pointer indifféremment sur une table, une variable ou un champ. Un même pointeur peut pointer sur des variables de type différent : veillez à ne pas générer des conflits artificiellement, en attribuant à une même variable des types différents.
Faites simplement attention à ne pas changer en cours de route le type de la variable sur laquelle pointe le pointeur en faisant, par exemple, une manipulation du type suivant :

```
LaVariable:=5,3
LePointeur:=-> LaVariable
LePointeur->:=6,4
LePointeur->:=Faux
```

Dans ce cas de figure, votre pointeur dépointé est une variable Numérique. En affectant à cette variable une valeur booléenne, vous provoquez un conflit de types.

Si vous avez besoin dans une même méthode d'utiliser les pointeurs pour des propos différents, prenez soin de définir votre pointeur :

```
LaVariable:=5,3
LePointeur:=-> LaVariable
LePointeur->:=6,4
LeBool:=Vrai
LePointeur:=->LeBool
LePointeur->:=Faux
```

Un pointeur n'a aucune existence propre. Il est toujours défini en fonction de l'objet qu'il représente. C'est pourquoi le compilateur ne peut pas détecter des conflits de types générés par une utilisation sans contrôle des pointeurs. Vous n'aurez donc pas, en cas de conflit, de message d'erreur durant la phase de typage ou celle de compilation.
Cela ne veut pas dire que lorsque vous utilisez des pointeurs, vous travaillez sans filet. Le compilateur peut vérifier vos utilisations de pointeurs lorsque vous cochez l'option **Contrôle d'exécution** dans les Préférences de compilation (cf. manuel *Mode Développement*).

Plug-ins

Généralités

Le compilateur a besoin, au moment de la compilation, des définitions des commandes des plug-ins utilisées dans la base à compiler, c'est-à-dire du nombre et du type des paramètres de ces commandes. Les risques d'erreur de typage sont inexistants à partir du moment où le compilateur trouve effectivement dans l'application ce que vous avez déclaré.

Assurez-vous que vos plug-ins sont installés dans le dossier PlugIns, à l'un des emplacements autorisés par 4D : à côté du fichier de structure de la base ou à côté de l'application exécutable (Windows) / dans le progiciel (Mac OS). Pour des raisons de compatibilité, il reste possible d'utiliser un dossier Win4DX ou Mac4DX à côté du fichier de structure. Pour plus d'informations, reportez-vous au *Guide d'installation* de 4D.
Le compilateur ne duplique pas le fichier mais tient compte des déclarations des commandes sans rien vous demander en supplément.
Si vos plug-ins sont placés ailleurs, le compilateur vous demande de les localiser lors du typage, via une boîte de dialogue d'ouverture de documents.

Routines recevant des paramètres implicites
Certains plug-ins, par exemple 4D Write, utilisent des commandes qui provoquent l'appel implicite à des commandes 4D.
Prenons un exemple avec 4D Write. Vous disposez d'une commande appelée WR APPELER SUR EVENEMENT. La syntaxe de cette commande est :
WR APPELER SUR EVENEMENT(LaZone;Evénement;MéthodeEvénement)

Le dernier paramètre que vous passez à cette routine est le nom d'une méthode, que vous aurez vous-même écrite dans 4D. Cette méthode sera appelée par 4D Write chaque fois que l'événement attendu sera reçu et cette méthode recevra automatiquement les paramètres suivants :

Paramètres	Type	Description
$0	Entier long	Retour de fonction
$1	Entier long	Zone 4D Write
$2	Entier long	Touche Maj.
$3	Entier long	Touche Alt (Windows), Option (Mac OS)
$4	Entier long	Touche Ctrl (Windows), Commande (Mac OS)
$5	Entier long	Type d'événement qui a provoqué l'appel
$6	Entier long	Valeur variant en fonction du paramètre Evénement

Afin que le compilateur connaisse l'existence de ces paramètres et les prenne en compte, vous devez vous assurer qu'ils peuvent effectivement être typés, soit par une directive de compilation, soit parce que leur utilisation, suffisamment explicite, permet de déduire leur type.

Composants 4D

4D permet de créer et de manipuler des composants. Un composant 4D est un ensemble d'objets 4D représentant une ou plusieurs fonctionnalité(s) groupée(s) dans un fichier de structure (appelé base matrice), qu'il est possible d'installer dans différentes bases (appelées bases hôtes).
Une base hôte exécutée en mode interprété peut utiliser indifféremment des composants interprétés ou compilés. Il est possible d'installer des composants interprétés et compilés dans la même base hôte. En revanche, une base hôte exécutée en mode compilé ne peut pas utiliser de composant interprété. Dans ce cas, seuls des composants compilés peuvent être employés.

Une base hôte interprétée contenant des composants interprétés peut être compilée si elle ne fait pas appel à des méthodes du composant interprété. Dans le cas contraire, une boîte de dialogue d'alerte apparaît lorsque vous tentez de compiler l'application et la compilation est impossible.

Un conflit de nom peut se produire lorsqu'une méthode projet partagée du composant a le même nom qu'une méthode projet de la base hôte. Dans ce cas, lorsque du code est exécuté dans le contexte de la base hôte, c'est la méthode de la base hôte qui est appelée. Ce principe permet de "masquer" une méthode du composant avec une méthode personnalisée (par exemple pour
obtenir une fonctionnalité différente). Lorsque le code est exécuté dans le contexte du composant, c'est la méthode du composant qui est appelée. Un masquage est signalé par un *warning* lors de la compilation de la base hôte.

Si deux composants partagent des méthodes du même nom, une erreur est générée au moment de la compilation de la base hôte.

Pour plus d'informations sur les composants, reportez-vous au manuel *Mode Développement*.

Manipulation des locales $0…$N et passation des paramètres

Les manipulations des variables locales suivent toutes les règles déjà énoncées. De même que toutes les autres variables, elles ne peuvent être retypées en cours de méthode.
Dans ce paragraphe, nous abordons deux cas de figure où l'inattention pourrait conduire à des conflits de types :
• Lorsque vous avez en fait besoin d'un retypage. L'utilisation de pointeurs vous permet alors d'éviter les conflits de types.
• Lorsque vous avez besoin d'adresser des paramètres par indirection.

Utiliser les pointeurs pour éviter les retypages

Il n'est pas possible de retyper une variable. Il est, en revanche, tout à fait possible de faire pointer un pointeur successivement sur des variables de type différent.
Un exemple nous permet d'illustrer ce propos : écrivons une fonction qui nous renvoie l'occupation mémoire d'un tableau à une dimension suivant son type. Le résultat est un numérique dans tous les cas sauf deux : dans le cas des tableaux Texte et des tableaux Image, la taille mémoire occupée par le tableau dépend de valeurs inexprimables sous forme numérique (cf. section Tableaux et mémoire).
Dans le cas des tableaux Texte et des tableaux Image, nous renverrons comme résultat une chaîne de caractères. Cette fonction requiert un paramètre : un pointeur sur le tableau dont on veut connaître l'occupation mémoire.
Pour effectuer cette opération, vous avez le choix entre deux méthodes :
• ne travailler qu'avec des variables locales et ne pas vous soucier de leur type — mais dans ce cas, la méthode ne fonctionnera qu'en mode interprété.
• utiliser des pointeurs et alors travailler indifféremment en interprété et en compilé.

• Fonction *OccupMém* en interprété seulement (exemple pour Macintosh)

```
$Taille:=Taille tableau($1->)
$Type:=Type($1->)
Au cas ou
   :($Type=Est un tableau numérique)
      $0:=8+($Taille*10)   ` $0 est un Numérique
   :($Type=Est un tableau entier)
      $0:=8+($Taille*2)
   :($Type=Est un tableau entierlong)
      $0:=8+($Taille*4)
   :($Type=Est un tableau date)
      $0:=8+($Taille*6)
   :($Type=Est un tableau texte)
      $0:=Chaine(8+($Taille*4))+("+Somme des longueurs des textes") ` $0 est un
                                                                              Texte
   :($Type=Est un tableau image)
      $0:=Chaine(8+($Taille*4))+("+Somme des tailles des images")   ` $0 est un Texte
   :($Type=Est un tableau pointeur)
      $0:=8+($Taille*16)
   :($Type=Est un tableau booléen)
      $0:=8+($Taille/8)
Fin de cas
```

Dans la méthode ci-dessus, il y a changement de type de $0 suivant les valeurs de $1.

• Fonction *OccupMém* en mode interprété et compilé (exemple pour Macintosh)
Ecrivons maintenant cette méthode en utilisant un pointeur :

```
$Taille:=Taille tableau($1->)
$Type:=Type($1->)
VarNum:=0
Au cas ou
   :($Type=Est un tableau numérique)
      VarNum:=8+($Taille*10)   ` VarNum est un Numérique
   :($Type=Est un tableau entier)
      VarNum:=8+($Taille*2)
   :($Type=Est un tableau entierlong)
      VarNum:=8+($Taille*4)
   :($Type=Est un tableau date)
      VarNum:=8+($Taille*6)
   :($Type=Est un tableau texte)
      VarText:=Chaine(8+($Taille*4))+("+Somme des longueurs des textes")
   :($Type=Est un tableau image)
      VarText:=Chaine(8+($Taille*4))+("+Somme des tailles des images")
```

```
    :($Type=Est un tableau pointeur)
        VarNum:=8+($Taille*16)
    :($Type=Est un tableau booléen)
        VarNum:=8+($Taille/8)
Fin de cas
Si (VarNum#0)
    $0:=->VarNum
Sinon
    $0:=->VarText
Fin de si
```

Il faut noter la différence entre ces deux séquences :
• dans le premier cas, le résultat de la fonction est la variable que l'on attendait,
• dans le second cas, le résultat de la fonction est un pointeur sur cette variable. Il vous appartient alors de simplement dépointer le résultat reçu.

Indirections sur les paramètres
Le compilateur gère la puissance et la souplesse de l'indirection sur les paramètres. En mode interprété, 4D vous donne toute latitude concernant le nombre et le type des paramètres. Vous gardez en mode compilé cette même liberté à condition de ne pas introduire de conflit de types et de ne pas utiliser, dans la méthode appelée, plus de paramètres que vous en avez passés, ce qui est facile.
Afin de contourner un éventuel conflit, les paramètres adressés par indirection doivent tous être du même type.
Pour une bonne gestion de cette indirection, il est important de respecter la convention suivante : si tous les paramètres ne sont pas adressés par indirection, ce qui est le cas le plus fréquent, il faut que les paramètres adressés par indirection soient passés en fin de liste.
A l'intérieur de la méthode, l'adressage par indirection se fait sous la forme : ${$i}, $i étant une variable numérique. ${$i} est appelé paramètre générique.

Illustrons notre propos par un exemple : écrivons une fonction qui prend des valeurs, fait leur somme et renvoie cette somme formatée suivant un format qui peut varier avec les valeurs.
A chaque appel à cette méthode, le nombre de valeurs à additionner peut varier. Il faudra donc passer comme paramètre à notre méthode les valeurs, en nombre variable, et le format, exprimé sous forme d'une chaîne de caractères.
Un appel à cette fonction se fera de la façon suivante :

```
Résultat:=LaSomme("##0,00";125,2;33,5;24)
```

La méthode appelante récupérera dans ce cas la chaîne : 182,70, somme des nombres passés, formatée suivant le format spécifié. D'après ce que l'on a vu plus haut, les paramètres de la fonction doivent être passés dans un ordre précis : le format d'abord et ensuite les valeurs, dont le nombre peut varier d'un appel à l'autre.

Examinons maintenant la fonction que nous appelons *LaSomme* :

```
$Somme:=0
Boucle($i;2;Nombre de parametres)
   $Somme:=$Somme+${$i}
Fin de boucle
$0:=Chaine($Somme;$1)
```

Cette fonction pourra être appelée de diverses manières :

```
Résultat:=LaSomme("##0,00";125,2;33,5;24)
Résultat:=LaSomme("000";1;18;4;23;17)
```

De même que pour les autres variables locales, la déclaration du paramètre générique par directive de compilation n'est pas obligatoire. Si elle est nécessaire (cas d'ambiguïté ou d'optimisation), elle se fait avec la syntaxe suivante :

```
C_ENTIER(${4})
```

La commande ci-dessus signifie que tous les paramètres à partir du quatrième (inclus) seront adressés par indirection. Ils seront tous de type Entier. Les types de $1, $2 et $3 pourront être quelconques. En revanche, si vous utilisez $2 par indirection, le type utilisé sera le type générique. Il sera donc de type Entier, même si pour vous, par exemple, il était de type Réel.

Note : Le compilateur utilisant cette commande durant le typage, le nombre, dans la déclaration, doit toujours être une constante et jamais une variable.

Variables réservées et constantes

Des variables et des constantes de 4D ont un type et une identité fixés par le compilateur. On ne peut donc créer une nouvelle variable, une méthode, une fonction ou une commande de plug-in portant le nom d'une de ces variables ou d'une de ces constantes. Bien entendu, vous pouvez tester leur valeur et les utiliser comme auparavant.

Variables système

Voici la liste complète des Variables système de 4D accompagnées de leur type.

Variable	Type
OK	Entier long
Document	Alpha (Chaîne fixe) : 255
FldDelimit	Entier long
RecDelimit	Entier long
Error	Entier long
MouseDown	Entier long
KeyCode	Entier long
Modifiers	Entier long

MouseX	Entier long
MouseY	Entier long
MouseProc	Entier long

Variables des états rapides
Lorsqu'on crée une colonne calculée dans un état, 4D crée automatiquement une variable C1 pour la première, C2, C3... pour les autres. Généralement, cette création est faite de façon transparente pour vous.
Dans le cas où vous utiliseriez ces variables dans des méthodes, souvenez-vous que, comme les autres variables, les variables C1, C2... ne peuvent être retypées.

Constantes prédéfinies 4D
La liste des constantes prédéfinies de 4D peut être consultée dans ce manuel, vous pouvez également les visualiser dans l'Explorateur, en mode Développement.

Référence

Conseils d'optimisation, Messages d'erreurs, Précisions de syntaxe, Utilisation des directives de compilation.

Le compilateur suit la syntaxe habituelle des commandes 4D et, en ce sens, ne vous demande aucun comportement particulier dans l'optique de la compilation.

Cette section propose cependant certains rappels et quelques précisions :
• Certaines commandes influant sur le type d'une variable peuvent, si vous n'y prêtez pas attention, conduire à des conflits de type.
• Certaines commandes admettant des syntaxes ou des paramètres différents, il est préférable de savoir ce qu'il est plus approprié de choisir.

Chaînes de caractères

Code de caractere (LaChaine)
Pour les routines opérant sur les chaînes, seule la fonction Code de caractere réclame une attention plus particulière. Lorsque vous travaillez en mode interprété, vous pouvez indifféremment passer une chaîne non vide ou vide à cette fonction.
En compilé, vous ne pouvez pas passer une chaîne vide.
Si vous le faites, le compilateur ne peut détecter une erreur à la compilation si l'argument passé à Code de caractere est une variable.

Communications

ENVOYER VARIABLE(LaVariable)
RECEVOIR VARIABLE(LaVariable)
Ces deux commandes permettent d'écrire et de relire des variables envoyées sur disque. On passe des variables en paramètres à ces commandes.
Souvenez-vous simplement qu'il faudra toujours relire une variable d'un type dans une variable de même type. Supposons que vous souhaitiez envoyer une liste de variables dans un fichier. Afin de ne pas prendre de risque de changement de type par inattention, nous vous recommandons d'adopter une méthode de travail simple qui consiste à indiquer en début de liste le type des variables envoyées. Ainsi, lorsque vous relirez ces variables, vous commencerez toujours par récupérer cet indicateur dont vous connaissez le type. Ensuite, vous appellerez RECEVOIR VARIABLE en toute connaissance de cause par l'intermédiaire d'un Au cas ou.

Exemple :

```
REGLER SERIE(12;"LeFichier")
   Si (OK=1)
      $Type:=Type([Client]CA_Cumulé)
            ENVOYER VARIABLE($Type)
```

```
            Boucle($i;1;Enregistrements trouves)
                $CA_Envoi:=[Client]CA_Cumulé
                ENVOYER VARIABLE($CA_Envoi)
            Fin de boucle
        Fin de si
        REGLER SERIE(11)
        REGLER SERIE(13;"LeFichier")
        Si (OK=1)
            RECEVOIR VARIABLE($Type)
            Au cas ou
                :($Type=Est une variable chaîne)
                    RECEVOIR VARIABLE($LaChaine)
                        `Traitement de la variable reçue
                :($Type=Est un numérique)
                    RECEVOIR VARIABLE($LeRéel)
                        `Traitement de la variable reçue
                :($Type=Est un texte)
                    RECEVOIR VARIABLE($LeTexte)
                        `Traitement de la variable reçue
            Fin de cas
        Fin de si
    REGLER SERIE(11)
```

Définition structure

Champ (Ptr_Champ) ***ou*** **(NoDeTable;NoDeChamp)**
Table(Ptr_Table) ***ou*** **(Ptr_Champ)** ***ou*** **(NoDeTable)**
Dans l'optique du compilateur, ces deux commandes ne comportent rien de spécifique. Nous appelons simplement votre attention sur un cas pratique : ces deux commandes retournent des résultats de type différent suivant le paramètre qui leur est passé :
• si vous passez un pointeur à la fonction Table, le résultat retourné sera de type Numérique.
• si vous passez un Numérique à la fonction Table, le résultat retourné sera de type Pointeur.
On comprend aisément que ces deux fonctions ne suffisent pas au compilateur pour déterminer le type du résultat. Dans ce cas, pour qu'il n'y ait aucune ambiguïté, utilisez une directive de compilation.

Documents système

Nous rappellerons simplement que la référence d'un document renvoyée par les fonctions Ouvrir document, Ajouter a document et Creer document est de type Heure.

Modulo (LaValeur;Diviseur)
L'expression "25 modulo 3" peut s'écrire de deux façons différentes dans 4D :

```
LaVariable:=Modulo(25;3)
```

ou

```
LaVariable:=25%3
```

Il existe pour le compilateur une différence entre ces deux écritures : Modulo s'applique à tous les types de Numériques tandis que l'opérateur % s'applique exclusivement aux Entiers et Entiers longs (si les opérandes de l'opérateur % dépassent les limites des Entiers longs, le résultat renvoyé sera probablement faux).

Interruptions

APPELER 4D
APPELER SUR EVENEMENT (LaMéthode {; NomProcess})
STOP

APPELER SUR EVENEMENT
Pour la gestion des interruptions, le langage de 4D dispose de la commande APPELER 4D. Cette commande devra être utilisée lorsque vous vous servirez de la commande APPELER SUR EVENEMENT.

On pourrait définir cette commande comme une directive de gestion des événements. Seul le noyau de 4D peut détecter un événement Système (clic souris, action sur le clavier…). Dans la plupart des cas, des appels au noyau sont lancés par le code compilé lui-même, de façon transparente pour vous.

En revanche, dans le cas où vous attendez un événement sans rien faire, comme dans une boucle d'attente, il est bien évident qu'aucun appel n'est effectué.

Exemple sous Windows

```
    `Méthode projet ClicSouris
Si (MouseDown=1)
    <>vTest:=Vrai
    ALERTE("Quelqu'un a cliqué avec la souris")
Fin de si

    `Méthode projet Attente
<>vTest:=Faux
APPELER SUR EVENEMENT("ClicSouris")
Tant que(<>vTest=Faux)
    `Boucle d'attente de l'événement
Fin tant que
APPELER SUR EVENEMENT("")
```

Dans ce cas, vous ajouterez la directive APPELER 4D de la façon suivante :

```
    `Méthode projet Attente
<>vTest:=Faux
APPELER SUR EVENEMENT("ClicSouris")
Tant que(<>vTest=Faux)
    APPELER 4D
        `Appel au noyau pour discerner un événement
Fin tant que
APPELER SUR EVENEMENT("")
```

STOP
Cette commande ne doit être utilisée que dans des méthodes projet d'interception d'erreurs. Cette commande fonctionne comme en mode interprété, sauf dans une méthode ayant été appelée par l'une des commandes suivantes : EXECUTER, APPLIQUER A SELECTION et APPLIQUER A SOUS SELECTION. Il convient d'éviter cette situation.

Tableaux

Sept routines de 4D sont utilisées par le compilateur pour déterminer le type d'un tableau. Il s'agit de :
COPIER TABLEAU(TableauSource;TableauDestination)
SELECTION VERS TABLEAU(LeChamp;LeTableau)
TABLEAU VERS SELECTION(LeTableau; LeChamp)
SELECTION LIMITEE VERS TABLEAU(Début;Fin;LeChamp;LeTableau)
ENUMERATION VERS TABLEAU(LaListe; LeTableau{;ItemRefs})
TABLEAU VERS ENUMERATION(LeTableau; LaListe{;ItemRefs})
VALEURS DISTINCTES(LeChamp;LeTableau)

COPIER TABLEAU
COPIER TABLEAU admet deux paramètres de type Tableau. Lorsque le compilateur rencontre cette commande pendant le typage et que l'un des paramètres tableau n'est pas déclaré ailleurs, le compilateur déduit le type du tableau non déclaré suivant le type de celui qui l'est.
Cette déduction est faite dans les deux cas suivants :
• le tableau typé ailleurs est le premier paramètre. Le compilateur donne au second tableau le type du premier.
• le tableau déclaré est le second paramètre. Dans ce cas, le compilateur donne au premier tableau le type du second.

Le compilateur étant rigoureux sur les types, un COPIER TABLEAU ne peut se faire que d'un tableau d'un type vers un tableau de même type.
En conséquence, si vous souhaitez faire une copie d'un tableau d'éléments de types voisins, c'est-à-dire les Entiers, Entiers longs et Numérique ou les Textes et les Alphas ou les Alphas de longueurs différentes, vous devrez le faire élément par élément.

Imaginez que vous vouliez faire une copie d'un tableau d'Entiers vers un tableau de Numériques. Procédez comme suit :

```
$Taille:=Taille tableau(TabEntier)
TABLEAU REEL(TabRéel;$Taille)
    `Donnons la même taille au tableau de Réels qu'au tableau d'Entiers
Boucle($i;1;$Taille)
    TabRéel{$i}:=TabEntier{$i}
        `Recopions chacun des éléments
Fin de boucle
```

Souvenez-vous que vous ne pouvez changer le nombre de dimensions d'un même tableau en cours de route. Vous provoqueriez une erreur de typage en copiant un tableau à une dimension dans un tableau à deux dimensions.

SELECTION VERS TABLEAU, TABLEAU VERS SELECTION, VALEURS DISTINCTES, SELECTION LIMITEE VERS TABLEAU

De même que pour 4D en mode interprété, ces quatre commandes n'exigent pas de déclaration de tableau. Le tableau non déclaré recevra le même type que le champ spécifié dans la commande.
Si vous écrivez :

```
SELECTION VERS TABLEAU([LaTable]ChampEntier;LeTableau)
```

le type de *LeTableau* sera donc Tableau d'Entiers à une dimension.

Dans le cas où le tableau a déjà été déclaré, veillez à ce que les champs soient du même type. Bien qu'Entier, Entier long et Réel soient des types voisins, vous ne pouvez pas supposer qu'il soient équivalents.
Vous avez en revanche plus de latitude lorsqu'il s'agit des types Texte et Alpha. Par défaut, lorsqu'un tableau n'a pas été déclaré au préalable et que vous appliquez l'une de ces commandes avec pour paramètre un champ de type Alpha, le type attribué au tableau sera Texte. Si le tableau a été déclaré auparavant comme étant de type Alpha ou de type Texte, ces commandes respecteront vos directives.

Il en est de même dans le cas des champs de type Texte : vos directives sont prioritaires.
Souvenez-vous que les commandes SELECTION VERS TABLEAU, SELECTION LIMITEE VERS TABLEAU, TABLEAU VERS SELECTION et VALEURS DISTINCTES ne s'appliquent qu'à des tableaux à une dimension.

La commande SELECTION VERS TABLEAU admet aussi une seconde syntaxe :

```
SELECTION VERS TABLEAU(LaTable;LeTableau)
```

Dans ce cas, la variable *LeTableau* sera de type Tableau d'Entiers longs. Il en est de même pour la commande SELECTION LIMITEE VERS TABLEAU.

ENUMERATION VERS TABLEAU, TABLEAU VERS ENUMERATION

Les commandes ENUMERATION VERS TABLEAU et TABLEAU VERS ENUMERATION ne concernent que deux types de tableaux :
• les tableaux Alpha à une dimension,

• les tableaux Texte à une dimension.
Ces deux commandes n'exigent pas la déclaration du tableau qui leur est passé en paramètre. Par défaut, un tableau non déclaré recevra le type Texte. Si le tableau a été déclaré auparavant comme étant de type Alpha ou de type Texte, ces commandes respecteront vos directives.

Utilisation des pointeurs dans les commandes s'appliquant aux tableaux
Le compilateur ne peut pas, durant le typage ou la compilation, détecter un conflit de type dans le cas où vous utiliseriez des pointeurs dépointés comme paramètre de commande de déclaration d'un tableau. Si vous écrivez :

SELECTION VERS TABLEAU([LaTable]LeChamp;LePointeur->)

où *LePointeur->* représente un tableau, le compilateur ne peut vérifier que le type du champ et du tableau sont identiques. Il vous appartient d'éviter alors ce type de conflit.

Pour cela, le compilateur vous fournit une aide précieuse : les **Warnings**. Chaque fois qu'il rencontre une routine de déclaration de tableau dans laquelle l'un des paramètres est un pointeur, il vous délivre un message vous invitant à la vigilance.

Tableaux locaux
Si vous souhaitez compiler une base de données qui utilise des tableaux locaux (tableaux visibles uniquement par les méthodes qui les ont créés), il est nécessaire de les déclarer explicitement dans 4D avant de les utiliser.

La déclaration explicite d'un tableau signifie l'utilisation d'une commande de type TABLEAU REEL, TABLEAU ENTIER, etc.

Par exemple, si une méthode génère un tableau local d'entiers contenant 10 valeurs, vous devez écrire au préalable la ligne suivante :

TABLEAU ENTIER($MonTableau;10)

Langage

Pointeur vers(NomVariable)
Type (Objet)
EXECUTER FORMULE(Instruction)
TRACE
PAS DE TRACE

Pointeur vers
Pointeur vers est une fonction qui retourne un pointeur sur le paramètre que vous lui avez passé. Supposons que vous vouliez initialiser un tableau de pointeurs. Chacun des éléments de ce tableau pointe vers une variable donnée. Ces variables sont au nombre de douze et nous les appelons V1, V2, …V12.

Vous pourriez écrire :

```
TABLEAU POINTEUR(Tab;12)
Tab{1}:=->V1
Tab{2}:=->V2
…
Tab{12}:=->V12
```

Vous pouvez aussi écrire :

```
TABLEAU POINTEUR(Tab;12)
Boucle($i;1;12)
   Tab{$i}:=Pointeur vers("V"+Chaine($i))
Fin de boucle
```

A la fin de l'exécution de cette séquence, vous récupérez un tableau de pointeurs dont chaque élément pointe sur une variable Vi.

Ces deux séquences sont bien entendu compilables. Toutefois, si les variables V1 à V12 ne sont pas utilisées explicitement ailleurs dans la base, le compilateur ne pourra pas les typer. Il vous faut donc les nommer ailleurs explicitement.
Cette déclaration explicite peut se faire de deux façons :
• soit en déclarant V1, V2, …V12 par une directive de compilation :

```
C_ENTIER LONG(V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12)
```

• soit en affectant ces variables dans une méthode :

```
V1:=0
V2:=0
…
V12:=0
```

Type (Objet)
Chaque variable de la base compilée n'ayant qu'un seul type, la fonction Type peut sembler d'un intérêt limité. Elle est cependant très précieuse lorsqu'on travaille avec des pointeurs. En effet, il peut être nécessaire de connaître le type de la variable pointée par un pointeur, la souplesse des pointeurs faisant que l'on ne sait pas toujours sur quoi ils pointent.

EXECUTER FORMULE
La commande EXECUTER FORMULE, historique dans 4D, présente des avantages en mode interprété qui n'ont pas grand sens en mode compilé.
En effet, en mode compilé, le nom de la méthode passé en paramètre à cette commande sera simplement interprété. Vous ne bénéficierez donc pas de tous les avantages apportés par le compilateur, sans compter que la syntaxe de votre paramètre ne pourra pas être vérifiée.
De surcroît, vous ne pouvez pas lui passer des variables locales comme paramètres.
On peut avantageusement remplacer un EXECUTER FORMULE par une série d'instructions. Nous vous en donnons ici deux exemples.

Soit la séquence suivante :

```
i:= FctFormulaire
EXECUTER FORMULE("FORMULAIRE ENTREE (Formulaire"+Chaine(i)+")")
```

Elle peut être remplacée par :

```
i:=FctFormulaire
VarFormulaire:="Formulaire"+Chaine(i)
FORMULAIRE ENTREE(VarFormulaire)
```

Examinons maintenant un deuxième cas :

```
$Num:=ChoixImprimante
EXECUTER FORMULE("Impri"+$Num)
```

Ici, l'EXECUTER peut être facilement remplacé par un Au cas ou :

```
Au cas ou
   : ($Num=1)
      Impri1
   : ($Num=2)
      Impri2
   : ($Num=3)
      Impri3
Fin de cas
```

La commande EXECUTER peut toujours être avantageusement remplacée. En effet, la méthode à exécuter est prise dans la liste des méthodes projet de la base ou des commandes 4D, qui sont en nombre limité. En conséquence, il est toujours possible de remplacer la commande EXECUTER soit par un Au cas ou, soit par une autre commande.

TRACE, PAS DE TRACE
Ces deux commandes, précieuses en mode interprété, n'ont pas de fonction en mode compilé. Vous pouvez cependant les laisser dans vos méthodes : TRACE et PAS DE TRACE seront simplement ignorées par le compilateur.

Variables

Indefinie(LaVariable)
ECRIRE VARIABLES(NomduDoc;Variable1{; Variable2…})
LIRE VARIABLES(NomduDoc;Variable1{; Variable2…})
Effacer variable(LaVariable)

Indefinie
Compte tenu du processus de typage par le compilateur, une variable ne peut à aucun moment être indéfinie en mode compilé. En effet, toutes les variables ont une existence dès la fin de la compilation. La fonction Indefinie retourne donc toujours Faux, quel que soit le paramètre que vous lui passez.

Note : Pour savoir si une application tourne en mode compilé, utilisez la fonction Mode compile.

ECRIRE VARIABLES, LIRE VARIABLES
En mode interprété, après l'exécution d'un LIRE VARIABLES, vous pouvez savoir si le document existe en testant si l'une des variables, théoriquement lue, est indéfinie ou non. Ceci n'est bien évidemment plus possible avec le compilateur, puisque la fonction Indefinie renvoie toujours Faux.

La méthode la plus simple pour réaliser cette opération en mode interprété comme en mode compilé est la suivante :
1. Initialisez les variables que vous allez recevoir à une valeur dont vous êtes sûr qu'elles ne sont pas initialisées.
2. Après le LIRE VARIABLES, comparez l'une des variables reçues à la valeur d'initialisation.
La méthode s'écrira alors de la façon suivante :

```
Var1:="xxxxxx"
    `"xxxxxx" est une valeur qui ne peut pas être ramenée par un LIRE VARIABLES
Var2:="xxxxxx"
Var3:="xxxxxx"
Var4:="xxxxxx"
LIRE VARIABLES("LeDocument";Var1;Var2;Var3;Var4)
Si(Var1="xxxxxx")
        `Le document n'a pas été trouvé
    …
Sinon
        `Le document a été trouvé
    …
Fin de si
```

EFFACER VARIABLE
Cette routine admet deux syntaxes différentes en mode interprété :
EFFACER VARIABLE(LaVariable)
EFFACER VARIABLE("a")
En mode compilé, la première syntaxe, EFFACER VARIABLE(LaVariable), provoque non la destruction de la variable, mais sa réinitialisation (remise à zéro pour un numérique, chaîne vide pour une chaîne de caractères ou un texte…), aucune variable ne pouvant être indéfinie en mode compilé.
En conséquence, EFFACER VARIABLE ne libère pas de mémoire en mode compilé sauf dans quatre cas : les variables de type Texte, Image, BLOB et les tableaux.
Pour un tableau, EFFACER VARIABLE équivaut à une nouvelle déclaration du tableau ou les tailles sont remises à zéro.

Pour un tableau *LeTableau* dont les éléments sont de type Entier, EFFACER VARIABLE(LeTableau) équivaut à l'une des expressions suivantes :

```
TABLEAU ENTIER(LeTableau;0)
    `s'il s'agit d'un tableau à une dimension
TABLEAU ENTIER(LeTableau;0;0)
    `s'il s'agit d'un tableau à deux dimensions
```

La seconde syntaxe, EFFACER VARIABLE("a"), n'est pas compatible avec le compilateur puisque celui-ci accède aux variables non par leur nom, mais par leur adresse mémoire.

Précisions concernant l'utilisation de pointeurs

Les commandes suivantes ont un point commun : elles admettent toutes un premier paramètre [LaTable] facultatif alors que le second paramètre peut être un pointeur.

ADJOINDRE ELEMENT
ALLER A ENREGISTREMENT
ALLER DANS SELECTION
APPLIQUER A SELECTION
CHARGER ENSEMBLE
CHERCHER
CHERCHER DANS SELECTION
CHERCHER PAR FORMULE
CHERCHER PAR FORMULE DANS SELECTION
COPIER SELECTION
DEPLACER SELECTION
DIALOGUE
EXPORTER TEXTE
ECRITURE DIF
ECRITURE SYLK
ENSEMBLE VIDE
QR ETAT
FORMULAIRE ENTREE
FORMULAIRE SORTIE
GRAPHE SUR SELECTION
IMPRIMER ETIQUETTES
Imprimer ligne
IMPORTER TEXTE
LECTURE DIF
LECTURE SYLK
LIEN RETOUR
NOMMER ENSEMBLE
REDUIRE SELECTION
ENLEVER ELEMENT
TRIER
TRIER PAR FORMULE
UTILISER PARAMETRES IMPRESSION
VERROUILLE PAR

Il est parfaitement possible en mode compilé de garder ce caractère optionnel au paramètre [LaTable]. Le compilateur fait toutefois une supposition dans le cas où le premier paramètre passé à l'une de ces commandes est un pointeur. Ne pouvant pas savoir sur quoi pointe ce pointeur, il suppose qu'il s'agit d'un pointeur de Table.

Prenons par exemple le cas de la commande CHERCHER dont la syntaxe est la suivante :
CHERCHER({LaTable{;LaFormule{;*}}})
Le premier élément du paramètre LaFormule doit être un champ.

Si vous écrivez :

CHERCHER(LePtrChamp->=**Vrai**)

le compilateur va chercher en deuxième élément de formule un symbole représentant un champ. Or, il trouvera le signe "=". Il délivrera un message d'erreur, ne pouvant identifier la commande avec une expression qu'il sait traiter.

En revanche, si vous écrivez :

CHERCHER(LePtrTable->;LePtrChamp->=**Vrai**)

ou

CHERCHER([LaTable];LePtrChamp->=**Vrai**)

vous levez toute ambiguïté.

Constantes

Si vous créez vos propres ressources 4DK# (constantes), assurez-vous que les numériques soient de type Entier long (L) ou Réel (R) et les alphas de type Chaîne (S). Tout autre type génèrera un Warning.

Référence

Conseils d'optimisation, Guide du typage, Messages d'erreurs, Utilisation des directives de compilation.

Il est difficile, voire impossible, de consigner une fois pour toutes des instructions pour "bien programmer". Tout au plus pouvons-nous renouveler les recommandations générales vous invitant à bien structurer vos programmes, grâce notamment aux possibilités de programmation générique offertes par 4D.
De fait, il est clair que si vous essayez de compiler une base très bien structurée, vous obtiendrez de bien meilleurs résultats que si votre base est mal structurée. Par exemple, si vous écrivez une méthode projet générique qui gère n méthodes objet, vous obtiendrez de bien meilleurs résultats, en interprété comme en compilé, que si les n méthodes objet comportent n fois les mêmes séquences d'instructions.
La qualité de votre programmation peut donc avoir des effets sur la qualité du code compilé. Si vous débutez, vous améliorerez progressivement votre code 4D grâce à l'habitude. De la même façon, c'est en utilisant fréquemment le compilateur que vous acquerrez les réflexes qui permettent inconsciemment d'aller droit au but.
En attendant cependant, voici quelques conseils et astuces qui vous permettront de gagner du temps dans des opérations simples et fréquentes.

Remarque préliminaire : les commentaires

Certaines astuces que nous vous conseillons peuvent rendre votre code moins lisible par une personne extérieure ou par vous-même ultérieurement. En conséquence, n'hésitez pas à assortir vos méthodes de commentaires détaillés. En effet, si les commentaires peuvent parfois ralentir une base interprétée, ils n'ont strictement aucune influence sur les temps d'exécution d'une base compilée.

Optimisation par les directives de compilation

Les directives de compilation peuvent vous aider à accélérer considérablement votre code. Par défaut, le compilateur donne le type le plus large possible à vos variables afin de ne pas vous pénaliser. Si vous ne typez pas par une directive de compilation une variable désignée par l'instruction Var:= 5, le compilateur lui donnera le type Réel, même s'il s'agit de l'affectation d'une valeur entière. Nous allons maintenant voir comment, dans certains cas, déclarer une variable peut vous faire gagner beaucoup de temps.

Numériques

Par défaut, le compilateur donne le type Réel (numérique) aux variables numériques non typées par directive de compilation et si les Préférences n'indique pas le contraire. Or, les calculs sur un Réel sont plus lents que sur un Entier. Lorsque vous êtes sûr qu'un de vos numériques s'exprimera toujours dans votre base sous forme d'un Entier, il est avantageux de le déclarer par les directives de compilation C_ENTIER ou C_ENTIER LONG.
Une bonne habitude à prendre, par exemple, est de systématiquement déclarer vos compteurs de boucles comme Entier par directive de compilation.

Par ailleurs, certaines fonctions standard de 4D renvoient des Entiers (exemple : Code de caractere, Ent...). Si vous affectez le résultat d'une de ces fonctions à une variable non typée de votre base, le compilateur lui donnera le type Numérique et non Entier. Pensez donc à déclarer ces variables par des directives de compilation si vous êtes certain qu'elles ne seront utilisées que dans ces conditions.

Prenons un exemple simple. Une fonction renvoyant une valeur aléatoire comprise entre deux bornes peut s'écrire :

```
$0:=Modulo(Hasard;($2-$1+1))+$1
```

Elle renvoie toujours une valeur entière. Si cette fonction est écrite ainsi, le compilateur donnera à $0 le type Numérique et non le type Entier ou Entier long. Il est donc préférable d'écrire cette méthode sous la forme :

```
C_ENTIER LONG($0)
$0:=Modulo(Hasard;($2-$1+1))+$1
```

Le paramètre rendu par la méthode sera plus petit et il sera donc plus rapide de faire appel à cette méthode.

Prenons un autre exemple simple. Supposons que vous ayez déclaré deux variables en Entier long par l'instruction suivante :

```
C_ENTIER LONG($var1;$var2)
```

et qu'une troisième variable non typée reçoive la somme des deux autres :

```
$var3:=$var1+$var2.
```

Il vous faudra explicitement déclarer *$var3* comme Entier long si vous voulez effectivement que *$var3* soit de type Entier long, sinon le compilateur la typera par défaut en Numérique.

Note : Attention au mode de calcul dans 4D. En mode compilé, ce n'est pas le type de la variable recevant le calcul qui détermine le mode de calcul, mais le type des opérandes.

• Dans l'exemple ci-dessous, le calcul s'effectue sur des Entiers longs :

```
C_REEL($var3)
C_ENTIER LONG($var1;$var2)
$var1:=2147483647
$var2:=1
$var3:=$var1+$var2
```

$var3 prendra la valeur –2147483648 en mode compilé comme en interprété.

• Dans l'exemple suivant :

```
C_REEL($var3)
C_ENTIER LONG($var1)
$var1:=2147483647
$var3:=$var1+1
```

pour des raisons d'optimisation, le compilateur considère la valeur 1 comme une valeur entière. *$var3* prendra alors la valeur –2147483648 en mode compilé (car le calcul s'effectue sur des Entiers longs) et 2147483648 en mode interprété (car le calcul s'effectue sur des Réels).

Les boutons sont un cas particulier de Numérique qu'il est possible de déclarer en Entier long.

Chaînes de caractères

De même que pour les numériques, le compilateur donne par défaut le type Texte à vos variables alphanumériques, si les Préférences n'indiquent pas le contraire. Si vous écrivez : MaChaine:="Bonjour", *MaChaine* sera pour le compilateur une variable de type Texte.
Si vous avez beaucoup de traitements à effectuer sur cette variable, il est avantageux de la déclarer explicitement à l'aide de la directive C_ALPHA. Les traitements sont en effet beaucoup plus rapides sur les variables de type Alpha dont la longueur maximale est définie, que sur les variables de type Texte. N'oubliez pas cependant les règles de comportement de cette directive.

Si vous souhaitez tester la valeur d'un caractère, il est plus rapide de faire la comparaison sur son Code de caractere que sur le caractère lui-même. En effet, la routine standard de comparaison de caractères prend en compte toutes les variations du caractère, comme par exemple les accentuations.

Remarques diverses

Tableaux à deux dimensions

Les traitements sur les tableaux à deux dimensions seront mieux gérés si la deuxième dimension est plus grande que la première.

Par exemple, un tableau déclaré sous la forme :

```
TABLEAU ENTIER(LeTableau;5;1000)
```

sera mieux géré qu'un tableau déclaré sous la forme :

```
TABLEAU ENTIER(LeTableau;1000;5)
```

Champs

Vous gagnerez du temps lorsque, si vous devez effectuer plusieurs calculs sur un champ, vous rangez la valeur de ce champ dans une variable et que vous effectuez vos calculs sur cette variable, plutôt que de faire directement les calculs sur le champ. Illustrons notre propos par un exemple. Prenons la méthode suivante :

```
Au cas ou
   : ([Client]Dest="Paris")
      Transport:="Coursier"
   : ([Client]Dest="Corse")
      Transport:="Avion"
   : ([Client]Dest="Etranger")
      Transport:="Service express"
   Sinon
      Transport:="Poste"
Fin de cas
```

La séquence ci-dessus sera plus lente à exécuter que si elle avait été écrite de la façon suivante :

```
$Dest:=[Client]Dest
Au cas ou
    : ($Dest="Paris")
        Transport:="Coursier"
    : ($Dest="Corse")
        Transport:="Avion"
    : ($Dest="Etranger")
        Transport:="Service express"
    Sinon
        Transport:="Poste"
Fin de cas
```

Pointeurs

De même que pour les champs, il est plus rapide de travailler sur une variable intermédiaire que sur un pointeur dépointé. Vous gagnerez énormément de temps si, lorsque vous devez faire plusieurs calculs sur une variable pointée, vous rangez le pointeur dépointé dans une variable.

Vous disposez par exemple d'un pointeur pointant sur un champ ou sur une variable, MonPtr. Ensuite, vous souhaitez faire une série de tests sur la valeur pointée par MonPtr. Vous pouvez écrire :

```
Au cas ou
    : (MonPtr-> = 1)
        Séquence 1
    : (MonPtr-> = 2)
        Séquence 2
    …
Fin de cas
```

Il est plus rapide d'exécuter cette série de tests si elle est écrite ainsi :

```
Temp:=MonPtr->
Au cas ou
    : (Temp = 1)
        Séquence 1
    : (Temp = 2)
        Séquence 2
    …
Fin de cas
```

Variables locales
Utiliser des variables locales permet, dans bien des cas, de mieux structurer son code. Ces variables présentent d'autres avantages :
• Les variables locales prennent moins de place en mémoire lors de l'utilisation d'une base : en effet, elles sont créées lorsqu'on entre dans la méthode où elles sont utilisées et détruites dès qu'on sort de cette méthode.
• Le code généré est optimisé pour les variables locales, en particulier de type Entier long. Pensez-y pour vos compteurs de boucle.

Référence

Guide du typage, Messages d'erreurs, Précisions de syntaxe, Utilisation des directives de compilation.

Cette section détaille les principaux messages délivrés par le compilateur. Ces messages sont de plusieurs types :
• les warnings, qui vous aident à déjouer des pièges facilement évitables ;
• les erreurs, qu'il vous appartient de corriger ;
• les messages de contrôle d'exécution, délivrés dans 4D.

Les warnings

Ces messages sont délivrés tout au long du processus de compilation. Chaque message est accompagné ici d'un exemple de ce qui a pu le provoquer et, éventuellement, d'une explication supplémentaire.

Utilisation de pointeur(s) comme paramètre(s) de COPIER TABLEAU
COPIER TABLEAU(LePointeur->;LeTableau)

Utilisation de pointeur(s) comme paramètre(s) de SELECTION VERS TABLEAU
SELECTION VERS TABLEAU(LePointeur->;LeTableau)
SELECTION VERS TABLEAU([LaTable]LeChamp;LePointeur->)

Utilisation de pointeur(s) comme paramètre(s) de TABLEAU VERS SELECTION
TABLEAU VERS SELECTION(LePointeur->;[LaTable]LeChamp)

Utilisation de pointeur(s) comme paramètre(s) de ENUMERATION VERS TABLEAU
ENUMERATION VERS TABLEAU(Enum;LePointeur->)

Utilisation de pointeur(s) comme paramètre(s) de TABLEAU VERS ENUMERATION
TABLEAU VERS ENUMERATION(LePointeur->;Enum)

Utilisation de pointeur(s) dans une déclaration de tableau
TABLEAU REEL(LePointeur->;5)
L'instruction TABLEAU REEL(LeTableau;LePointeur->) ne provoquera pas cet avertissement. La valeur de la dimension d'un tableau n'a pas d'influence sur son type.

Utilisation de pointeur(s) comme paramètre(s) de VALEURS DISTINCTES
VALEURS DISTINCTES(LePointeur->;LeTableau)

Utilisation de la fonction Indefinie
Si(**Indefinie**(LaVariable))

Cette méthode est protégée par un mot de passe.

Le Formulaire LeFormulaire contient un bouton avec action sans nom dans la page 1.
Tous vos boutons avec action doivent avoir un nom afin d'éviter des conflits.

Le pointeur utilisé dans cette commande doit pointer sur un Alphanumérique.

```
LePointeur->[[2]]:="a"
```

Le pointeur utilisé dans cette commande doit pointer sur un Entier, un Entier long ou un Numérique

```
LaChaine[[LePointeur->]]:="a"
```

Un indice de tableau doit être Entier, Entier long ou Numérique.

```
ALERTE(LeTableau{LePointeur->})
```

Il manque un paramètre à l'appel de la commande du plug-in.

```
WR FIXER POLICE(LaZone)
```

Les erreurs

Ces messages vous sont délivrés tout au long du processus de compilation. Il vous appartient de corriger ces erreurs afin de permettre au compilateur de générer une base compilée. Chaque message est accompagné ici d'un exemple de ce qui a pu le provoquer et éventuellement, d'une explication supplémentaire.
Les messages sont classés thématiquement. Les thèmes dans lesquels vous pourrez les consulter sont Typage, Syntaxe, Paramètres, Opérateurs, Plug-ins et Erreurs générales.

• Typage

Cet opérateur ne peut s'appliquer à ce type de variable. Cette affectation provoquerait un conflit de type.

```
LeRéel:=12,3
LeBooléen:=Vrai
LeRéel:=LeBooléen
```

Changement de la longueur maximale d'une chaîne de caractères.

```
C_ALPHA(3;LaChaine)
C_ALPHA(5;LaChaine)
```

Changement du nombre de dimensions d'un tableau.

```
TABLEAU TEXTE(LeTableau;5;5)
TABLEAU TEXTE(LeTableau;5)
```

Conflit de type sur la variable LeTableau dans le formulaire LeFormulaire.

```
TABLEAU ENTIER(LeTableau)
```

Déclaration d'un tableau sans indice.

```
TABLEAU ENTIER(LeTableau)
```

Il manque une variable.

```
COPIER TABLEAU(LeTableau;"")
```

Il manque une constante.

```
C_ALPHA(LaVariable;LaChaine)
```

Impossible de déterminer le type de LaVariable. Cette variable est utilisée dans la méthode M1.
Le type de *LaVariable* n'a pu être déterminé. Une directive de compilation est nécessaire.

Type de constante invalide

```
OK:="Il fait beau"
```

La méthode M1 est inconnue.
La ligne contient un appel à une méthode qui n'existe pas ou plus.

Le champ utilisé dans cette expression provoque un conflit de type.

```
MaDate:=Ajouter a date(LeChampBool;1;1;1)
```

La taille d'une chaîne de caractères ne peut pas dépasser 255.

```
C_ALPHA(325;LaChaine)
```

La variable LaVariable n'est pas une méthode.

```
LaVariable(1)
```

La variable LaVariable n'est pas un tableau.

```
LaVariable{5}:=12
```

Le résultat de la fonction est incompatible avec l'expression.

```
LeTexte:="Numéro"+Num(i)
```

Les types utilisés dans l'expression sont incompatibles.

```
LEntier:=LaDate*LeTexte
```

Utilisation de la variable $i de type alphanumérique comme variable de type réel.

```
$i:="3"
$($i):=5
```

L'indice du tableau n'est pas numérique.

```
TabEntier{"3"}:=4
```

Retypage de la variable LaVariable du type Texte en tableau de type Texte.

```
C_TEXTE(LaVariable)
COPIER TABLEAU(TabTexte;LaVariable)
```

Retypage de la variable LaVariable du type Texte en type Réel.

```
LaVariable:=Num(LaVariable)
```

Retypage du tableau LeBooléen du type Booléen en variable de type Réel.

```
LaVariable:=LeBooléen
```

Retypage du tableau du type Entier en type tableau de type Texte.

```
TABLEAU TEXTE(TabEntier;12)
```

si TabEntier a été déclaré ailleurs comme tableau d'Entiers.

Seuls les pointeurs peuvent être suivis du signe ->

```
LaVariable->:=5
```

si LaVariable n'est pas de type Pointeur.

Utilisation de la variable LaVar1 de type Texte comme une variable de type Numérique.

```
LaVar1:=3,5
```

Utilisation d'un champ de type incorrect. [LaTable]LeChamp est un champ de type Date.

LaVariable est de type Numérique.

```
LaVariable:=[LaTable]LeChamp
```

• Syntaxe

Cette fonction ne retourne pas un pointeur.

```
LaVariable:=Num("Il fait beau")->
```

Il n'est pas possible de dépointer cette fonction.

Erreur de syntaxe.

```
Si(LeBooléen)
Fin de boucle
```

Cette expression contient trop d'accolades ouvrantes ({).
La ligne comporte plus d'accolades ouvrantes que d'accolades fermantes.

Cette expression contient trop d'accolades fermantes (}).
La ligne comporte plus d'accolades fermantes que d'accolades ouvrantes.

Il manque une parenthèse fermante.
La ligne comporte plus de parenthèses ouvrantes que de fermantes.

Il manque une parenthèse ouvrante.
La ligne comporte plus de parenthèses fermantes que d'ouvrantes.

J'attendais un champ.

```
Si(Modifie(LaVariable))
```

Il manque une accolade ouvrante.

```
C_ENTIER($
```

J'attendais une variable.
```
C_ENTIER([LaTable]LeChamp)
```

J'attendais un nombre constant.
```
C_ENTIER(${"3"})
```

J'attendais un point virgule.
```
COPIER TABLEAU(LeTableau1 LeTableau2)
```

• Mac OS :
Cette expression contient trop d'indices de chaîne ouvrants.
```
LaChaine[[3:="a"
```
Cette expression contient trop d'indices de chaîne fermants.
```
LaChaine3]]:="a"
```

• Windows :
Cette expression contient trop d'indices de chaîne ouvrants.
```
LaChaine[[3:="a"
```
Cette expression contient trop d'indices de chaîne fermants.
```
LaChaine 3]]:="a"
```

Je n'attendais pas un champ de type sous-table
```
TABLEAU VERS SELECTION(LeTableau;Soustable)
```

Le type du paramètre de SI doit être booléen.
```
Si(LePointeur)
```

L'expression est trop complexe.
Divisez votre ligne en plusieurs sous-opérations.

Méthode trop complexe.
Trop d'imbrications de Au cas ou...Fin de cas et de Si…Fin de si.

Référence à un champ inconnu.
Votre méthode, probablement copiée d'une autre base, contient •???• à la place d'un nom de champ.

Référence à une table inconnue.
Votre méthode, probablement copiée d'une autre base, contient •???• à la place d'un nom de table.

Un pointeur ne peut être défini sur cette expression.
```
LePointeur:=->LaVariable+3
```

Utilisation incorrecte des indices de chaînes de caractères.

```
LeNumérique[[3]] ou LeNumérique [[3]]
```

ou bien

```
LaChaine[[LaVariable]] ou LaChaine[[LaVariable]]
```

où *LaVariable* n'est pas une variable Numérique.

• Paramètres

Ce résultat de fonction ne peut pas être paramètre de cette méthode.

```
LaMéthode(Num(LaChaine))
```

si LaMéthode attend un paramètre de type Booléen.

Cette routine reçoit trop de paramètres

```
TABLE PAR DEFAUT(LaTable;LeFormulaire)
```

Cette valeur ne peut pas être paramètre de cette méthode.

```
LaMéthode(3+2)
```

si LaMéthode attend un paramètre de type Booléen.

Conflit de type sur la variable $0.

```
C_ENTIER($0)
$0:=Faux
```

Conflit de type sur le paramètre générique.

```
C_ENTIER(${3})
Boucle($i;3;5)
   ${$i}:=Chaine($i)
Fin de boucle
```

La routine n'attend pas de paramètre.

```
AFFICHER BARRE OUTILS(MaVar)
```

La routine nécessite au moins un paramètre.

```
TABLE PAR DEFAUT
```

La variable LaChaine ne peut pas être paramètre de cette méthode.

```
LaMéthode(LaChaine)
```

si LaMéthode attend un paramètre de type Booléen.

Le type du paramètre $1 dans la méthode est différent de celui du paramètre à l'appel.

```
Calcul("3+2")
```

avec la directive C_ENTIER($1) dans *Calcul*.

Le type du paramètre passé ne correspond pas au type du paramètre attendu.

```
Impri("LaserWriter")
```

si dans la méthode *Impri*, $1 est de type Numérique.

L'un des paramètres de COPIER TABLEAU est une variable.

```
COPIER TABLEAU(LaVariable;LeTableau)
```

Retypage du paramètre $1 du type réel en type Texte.

```
$1:=Chaine($1)
```

Un paramètre ne peut être un tableau.

```
Rélnit(LeTableau)
```

Pour passer un tableau à une méthode, il faut passer un pointeur sur ce tableau.

Un paramètre ne peut être utilisé dans l'appel de cette routine.

```
RECEVOIR VARIABLE($1)
```

Utilisation du paramètre $1 de type Booléen comme une variable de type Entier.

```
LIRE PROPRIETES CHAMP(NoDeTable;NoDeChamp;Type;$1)
```

• Opérateurs

Cet opérateur ne peut s'appliquer à ce type de variable.

```
LeBool2:=LeBool1+Vrai
```

L'addition ne peut pas s'appliquer à des Booléens.

Je n'attendais pas l'opérateur >

```
CHERCHER(LaTable;[LaTable]LeChamp=0;>)
```

Les deux opérandes ne sont pas comparables.

```
Si(LeLongE=Image2)
```

Le signe moins ne peut pas être utilisé dans cette expression.

```
LeBool:=-Faux
```

• Plug-ins

La commande PExt du plug-in est mal définie.

Il manque des paramètres à l'appel de la commande du plug-in.

Le nombre de paramètres envoyés à la commande du plug-in est trop grand.

La commande LaVariable du plug-in est mal définie.

• Erreurs générales

Deux méthodes portent le même nom : LeNom.
Pour pouvoir compiler votre base, il faut que toutes les méthodes projets aient des noms différents.

Erreur interne n° xx.
Au cas où ce message apparaîtrait dans l'une de vos bases, téléphonez au support technique de 4D et signalez le numéro de l'erreur.

Je n'ai pas pu déterminer le type de LaVariable. Cette variable est utilisée dans la méthode M1.
Le type de LaVariable n'a pu être déterminé. Une directive de compilation est nécessaire.

La méthode originale est endommagée.
La méthode est endommagée dans la structure originale. Supprimez-la ou remplacez-la.

Méthode 4D inconnue.
La méthode est endommagée.

Retypage de la variable LaVariable dans le Formulaire LeFormulaire.
Ce message apparaîtra si vous donnez, par exemple, le nom *OK* à une variable de type Graphe dans un formulaire.

Une fonction et une variable portent le même nom : LeNom.
Renommez soit la fonction, soit la variable.

Une variable dessinée dans le Formulaire LeFormulaire a le même nom qu'une fonction : LeNom.
Renommez soit la fonction, soit la variable.

Une méthode et une variable portent le même nom : LeNom.
Renommez soit la méthode, soit la variable.

Une commande du plug-in et une variable portent le même nom : LeNom.
Renommez soit la commande du plug-in, soit la variable.

Les messages du contrôle d'exécution

Ces messages sont délivrés dans 4D, lors de l'utilisation de la base compilée. Ils sont affichés dans 4D dans une fenêtre d'erreur spécifique.

Dépassement de capacité d'un tableau.
Si LeTableau est un tableau à 5 éléments à un instant donné, ce message apparaîtra si vous essayez d'accéder à l'élément LeTableau{17} à cet instant.

Division par zéro.

```
Var1:=0
Var2:=2
Var3:=Var2 / Var1
```

Le paramètre utilisé n'a pas été passé.
Utilisation de la variable $4 alors que seuls trois paramètres ont été passés lors de l'appel courant.

Le pointeur n'est pas correctement initialisé.

```
LePointeur->:=5
```

si *LePointeur* n'a pas encore été initialisé.

La chaîne dans laquelle se fait l'affectation est trop courte.

```
C_ALPHA(LaChaine1;5)
C_ALPHA(LaChaine2;10)
LaChaine2 :="Bonjour"
LaChaine1:= LaChaine2
```

L'indice de chaîne n'est pas valide (trop grand ou négatif).

```
i:=-30
```

LaChaine[[i]]:= LaChaine2 ou LaChaine[[i]]:=LaChaine2

La chaîne passée en paramètre est vide ou non initialisée.

```
LaChaine[[1]]:= ""
LaChaine[[1]]:= ""
```

Modulo par zéro.

```
Var1:=0
Var2:=2
Var3:=Var2 % Var1
```

Paramètres incorrects dans une commande EXECUTER FORMULE.

```
EXECUTER FORMULE("MaMéthode(MonAlpha)")
```

si *MaMéthode* attend un paramètre autre qu'alphanumérique.

Pointeur sur une variable inconnue du compilateur.

```
LePointeur:= Pointeur vers ("LaVariable")
LePointeur:= "MaChaîne"
```

si *LaVariable* n'apparaît pas explicitement dans la base.

Tentative de retypage par l'intermédiaire d'un pointeur.

```
LeBooleen:=LePointeur->
```

si *LePointeur* référence un champ de type Entier.

Utilisation incorrecte d'un pointeur.

```
LeCaractère:=LaChaine [[LePointeur->]]
LeCaractère:=LaChaine[[LePointeur]]
```

si *LePointeur* ne référence pas un Numérique.

Référence

Conseils d'optimisation, Guide du typage, Précisions de syntaxe, Utilisation des directives de compilation.

APPELER 4D

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

APPELER 4D est destinée uniquement à une utilisation avec le compilateur. En effet, seul le moteur de 4D peut détecter un événement. Il était donc nécessaire, dans le cadre d'une base compilée, qu'une routine puisse interroger le moteur de 4D afin de savoir si un événement s'est produit. Cette commande doit donc être utilisée lorsque vous employez la commande APPELER SUR EVENEMENT.
Par exemple, si une méthode exécute une boucle dans laquelle aucune commande 4D n'est appelée, la boucle ne pourra pas être interrompue par un process installé à l'aide d'APPELER SUR EVENEMENT, et l'utilisateur ne pourra pas ouvrir une autre application. Dans ce cas, APPELER 4D doit être insérée pour que 4D puisse intercepter les événements. Bien entendu, n'utilisez pas APPELER 4D si vous ne voulez aucune interruption.

Exemple

Dans l'exemple suivant, la boucle ne se terminerait jamais dans une base compilée sans l'aide de APPELER 4D :

```
    ` Méthode Traitement quelconque
APPELER SUR EVENEMENT ("METHODE EVENEMENT")
<>vbArrêt:=Faux
MESSAGE ("Traitement..."+Caractere(13)+"Tapez une touche pour interrompre
                                                    l'exécution...")
Repeter
        ` Effectuer un traitement sans appel à une commande 4D
    APPELER 4D
Jusque (<>vbArrêt)
APPELER SUR EVENEMENT ("")
```

La méthode METHODE EVENEMENT :

```
    ` Méthode METHODE EVENEMENT
Si (Indefinie(Keycode))
    Keycode:=0
Fin de si
Si (Keycode#0)
    CONFIRMER ("Voulez-vous vraiment interrompre cette opération ?")
    Si (OK=1)
        <>vbArrêt:=Vrai
    Fin de si
Fin de si
```

Référence

APPELER SUR EVENEMENT, Commandes du thème Compilateur.

C_ALPHA ({méthode; }taille; variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
taille	Numérique	→	Taille de la chaîne
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_ALPHA affecte le type Alphanumérique à chaque variable spécifiée.

Le paramètre taille spécifie la longueur maximum des chaînes qu'une variable peut contenir. En mode non Unicode (compatibilité ASCII), les chaînes sont limitées à 255 caractères. Si vous souhaitez optimiser la vitesse d'exécution de votre base, il est préférable d'utiliser des variables Alpha au lieu de variables Texte lorsque c'est possible.

Note de compatibilité : Les bases de données créées à compter de la version 11 de 4D sont exécutées par défaut en mode Unicode (cf. section Codes ASCII). Dans ce mode, le fonctionnement de la commande C_ALPHA est rigoureusement identique à la celui de commande C_TEXTE (le paramètre taille est ignoré). Il est conseillé d'utiliser C_TEXTE dans les nouveaux développements. La commande C_ALPHA est conservée pour des raisons de compatibilité uniquement.

La première syntaxe de la commande (lorsque le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_ALPHA(...;${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_ALPHA(...;${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, C_TEXTE, Nombre de parametres.

C_BLOB ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_BLOB assigne le type BLOB à toutes les variables spécifiées.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_BLOB(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_BLOB(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous aux exemples de la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur.

C_BOOLEEN ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_BOOLEEN affecte le type Booléen à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_BOOLEEN(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_BOOLEEN(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, Nombre de parametres.

C_DATE ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_DATE affecte le type Date à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_DATE(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_DATE(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, Nombre de parametres.

C_ENTIER ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Note préliminaire : La commande C_ENTIER est conservée pour des raisons de compatibilité avec les anciennes bases de données. En effet, en interne 4D et le compilateur retypent les Entiers en Entiers longs. Par exemple :
C_ENTIER($MaVar)
$Letype:=**Type**($MaVar) `$Letype vaut 9 (Est un entier long)

Description

C_ENTIER affecte le type Entier à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Exemple

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, C_ENTIER LONG, C_REEL, Nombre de parametres.

C_ENTIER LONG ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_ENTIER LONG affecte le type Entier long à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_ENTIER LONG(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_ENTIER LONG(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, C_ENTIER, C_REEL, Nombre de parametres.

C_GRAPHE ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Alpha	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_GRAPHE affecte le type Graphe à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_GRAPHE(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_GRAPHE(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Référez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur.

C_HEURE ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_HEURE affecte le type Heure à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_HEURE(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_HEURE(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, Nombre de parametres.

C_IMAGE ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_IMAGE affecte le type Image à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_IMAGE(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_IMAGE(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, Nombre de parametres.

C_POINTEUR ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_POINTEUR affecte le type Pointeur à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_POINTEUR(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_POINTEUR(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, Nombre de parametres.

C_REEL ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_REEL affecte le type Réel (numérique) à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_REEL(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_REEL(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, C_ENTIER, C_ENTIER LONG, Nombre de parametres.

C_TEXTE ({méthode; }variable{; variable2; ...; variableN})

Paramètre	Type		Description
méthode	Méthode	→	Nom de méthode
variable	Variable ou ${...}	→	Nom(s) de(s) variable(s) à déclarer

Description

C_TEXTE affecte le type Texte à chaque variable spécifiée.

La première syntaxe de la commande (si le paramètre méthode n'est pas passé) est utilisée pour déclarer et typer une variable process, interprocess ou locale.

Note : Cette syntaxe peut être utilisée dans les bases interprétées.

La seconde syntaxe de la commande (si le paramètre méthode est passé) est utilisée pour déclarer d'avance pour au compilateur le résultat et/ou les paramètres ($0, $1, $2, etc.) d'une méthode. Vous devez utiliser cette syntaxe si vous voulez éviter la phase de typage de variables lors de la compilation d'une base, afin de réduire le temps de compilation.

ATTENTION : La deuxième syntaxe ne peut pas être exécutée en mode interprété. Pour cette raison, si vous utilisez cette syntaxe, stockez-la dans une méthode (dont le nom doit débuter par "COMPILER") qui n'est pas exécutée en mode interprété.

Utilisation avancée : La syntaxe C_TEXTE(${...}) vous permet de déclarer pour une méthode un nombre variable de paramètres du même type à la condition que ce soient les derniers paramètres de la méthode. Par exemple, la déclaration C_TEXTE(${5}) indique à 4D et au compilateur qu'à partir du cinquième paramètre la méthode peut recevoir un nombre variable de paramètres de ce type. Pour plus d'informations, référez-vous à la commande Nombre de parametres.

Exemples

Reportez-vous à la section Commandes du thème Compilateur.

Référence

Commandes du thème Compilateur, C_ALPHA, Nombre de parametres.

8

Conteneur de données

Les commandes du thème “Conteneur de données” permettent de prendre en charge à la fois les actions liées au copier-coller (gestion du Presse-papiers) ainsi que celles liées au glisser-déposer inter-applications.

4D exploite deux conteneurs de données : l’un pour les données copiées (ou coupées), qui est en fait celui du Presse-papiers, déjà présent dans les versions précédentes de 4D et l’autre pour les données en cours de glisser-déposer.
Ces deux conteneurs sont gérés à l’aide des mêmes commandes. Vous accédez à l’un ou à l’autre en fonction du contexte :
• Le conteneur de glisser-déposer est accessible uniquement dans le cadre des événements formulaire Sur début glisser, Sur glisser ou Sur déposer et dans la Méthode base Sur déposer. En-dehors de ces contextes, le conteneur de glisser-déposer n’est pas disponible.
• Le conteneur de copier-coller est accessible dans tous les autres cas. A la différence du conteneur de glisser-déposer, il conserve durant la session les données qui y ont été placées, tant qu’il n’a pas été effacé ou réutilisé.

Types de données

Lors du glisser-déposer, différents types de données peuvent être placés et lus dans le conteneur de données. Vous pouvez accéder à un type de données de plusieurs manières :
• via sa **signature 4D** : la signature 4D est une chaîne de caractères indiquant un type de données référencé par 4D. L’emploi de signatures 4D facilite le développement d’application multi plates-formes, car ces signatures sont identiques sous Mac OS et Windows. Vous trouverez ci-dessous la liste des signatures 4D.
• via un **UTI** (*Uniform Type Identifier*, Mac OS uniquement) : la norme UTI, définie par Apple, associe une chaîne de caractères à chaque type d’objet natif. Par exemple, les images GIF ont le type UTI “com.apple.gif”. Les types UTI sont publiés dans les documentations Apple ainsi que par les éditeurs concernés.
• via son **numéro** ou **son nom de format** (Windows uniquement) : sous Windows, chaque type de donnée natif est référencé par un numéro (“3”, “12”, etc.) et un nom (“Rich Text Edit”). Par défaut, Microsoft définit plusieurs types natifs appelés formats de données standard. En outre, tout éditeur tiers peut “enregistrer” des noms de formats auprès du système, qui leur attribue un numéro en retour. Pour plus d’informations sur ce principe et sur les types natifs, reportez-vous à la documentation développeur de Microsoft (en particulier http://msdn2.microsoft.com/en-us/library/ms649013.aspx).

Note : Dans les commandes de 4D, les numéros de formats Windows sont manipulés sous forme de textes.

Toutes les commandes du thème “Conteneur de données” peuvent travailler avec chacun de ces types de données. Vous pouvez connaître les types de données présents dans le conteneur dans chacun de ces formats à l’aide de la commande LIRE TYPE DONNEES DANS CONTENEUR.

Note : Les types sur 4 caractères (TEXT, PICT ou types personnalisés) sont conservés par compatibilité avec les versions précédentes de 4D.

Signatures 4D

Voici la liste des signatures 4D standard ainsi que leur description :

Signature	Description
"com.4d.private.text.native"	Texte en jeu de caractères natif
"com.4d.private.text.utf16"	Texte en jeu de caractères unicode
"com.4d.private.text.rtf"	Texte enrichi
"com.4d.private.picture.pict"	Image format PICT
"com.4d.private.picture.png"	Image format PNG
"com.4d.private.picture.gif"	Image format GIF
"com.4d.private.picture.jfif"	Image format JPEG
"com.4d.private.picture.emf"	Image format EMF
"com.4d.private.picture.bitmap"	Image format BITMAP
"com.4d.private.picture.tiff"	Image format TIFF
"com.4d.private.picture.pdf"	Document PDF
"com.4d.private.file.url"	Chemin d’accès de fichier

AJOUTER DONNEES AU CONTENEUR (typeDonnées; données)

Paramètre	Type		Description
typeDonnées	Alpha	→	Type des données à ajouter
données	BLOB	→	Données à ajouter au conteneur

Description

AJOUTER DONNEES AU CONTENEUR ajoute dans le conteneur les données du type spécifié dans typeDonnées présentes dans le BLOB données.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Passez dans typeDonnées une valeur définissant le type de données à ajouter. Vous pouvez passer une signature 4D, un type UTI (Mac OS), un nom/numéro de format (Windows), ou un type de 4 caractères (compatibilité). Pour plus d'informations sur ces types, reportez-vous à la section Gestion du conteneur de données.

Généralement, vous utilisez la commande AJOUTER DONNEES AU CONTENEUR pour placer plusieurs instances des mêmes données dans le conteneur de données ou pour y ajouter des valeurs qui ne sont pas du texte ou une image. Pour ajouter de nouvelles données au conteneur, il faut d'abord l'effacer à l'aide de la commande EFFACER CONTENEUR.

Si vous voulez effacer le conteneur et y ajouter :
• du texte, utilisez la commande FIXER TEXTE DANS CONTENEUR,
• une image, utilisez la commande FIXER IMAGE DANS CONTENEUR,
• un chemin d'accès de fichier (glisser-déposer), utilisez la commande FIXER FICHIER DANS CONTENEUR.

Notez cependant que si un BLOB contient du texte ou une image, vous pouvez utiliser la commande AJOUTER DONNEES AU CONTENEUR pour y ajouter du texte ou une image.

Exemple

A l'aide des commandes du thème Conteneur de données et des BLOBs, vous pouvez écrire des méthodes de Couper/Copier/Coller pour gérer des données structurées au lieu d'une seule information. Dans l'exemple suivant, les deux méthodes projet écrire enregistrement dans Presse-papiers et lire enregistrement dans Presse-papiers vous permettent de traiter un enregistrement comme une information à copier dans le Presse-papiers.

```
` Méthode projet écrire enregistrement dans Presse-papiers
` écrire enregistrement dans Presse-papiers ( Numérique )
` écrire enregistrement dans Presse-papiers ( Numéro de table )

C_ENTIER LONG($1;$vlChamp;$vlTypeChamp)
C_POINTEUR($vpTable;$vpChamp)
C_ALPHA(255;$vaNomDoc)
C_TEXTE($vtDonnéesEnregistrement;$vtDonnéesChamp)
C_BLOB($vxDonnéesEnregistrement)

   ` Effacer le Presse-papiers (il restera vide s'il n'y a pas d'enregistrement courant)
EFFACER CONTENEUR
   ` Obtenir un pointeur vers la table dont le numéro est passé en paramètre
$vpTable:=Table($1)
   ` S'il y a un enregistrement courant pour cette table
Si ((Numero enregistrement($vpTable->)>=0) | (Nouvel enregistrement($vpTable->)))
      ` Initialiser la variable Texte qui contiendra l'image de texte de l'enregistrement
   $vtDonnéesEnregistrement:=""
      ` Pour chaque champ de l'enregistrement :
   Boucle ($vlChamp;1;Lire numero dernier champ($1))
         ` Obtenir le type du champ
      LIRE PROPRIETES CHAMP($1;$vlChamp;$vlTypeChamp)
         ` Obtenir un pointeur vers le champ
      $vpChamp:=Champ($1;$vlChamp)
         ` Selon le type du champ, copier (ou non) ses données de façon appropriée
      Au cas ou
         : (($vlTypeChamp=Champ alphanumérique ) | ($vlTypeChamp=Texte ))
            $vtDonnéesChamp:=$vpChamp->
         : (($vlTypeChamp=Numérique ) | ($vlTypeChamp=Entier ) | ($vlTypeChamp=
                                                                  Entier long)
                                    | ($vlTypeChamp=Champ ou variable date )
                                    | ($vlTypeChamp=Champ ou variable heure ))
            $vtDonnéesChamp:=Chaine($vpChamp->)
```

```
            : ($vlTypeChamp=Booléen )
                $vtDonnéesChamp:=Chaine(Num($vpChamp->);"Oui;;Non")
        Sinon
                ` Passer et ignorer les autres types de champs
            $vtDonnéesChamp:=""
        Fin de cas
            ` Accumuler les données sur le champ dans une variable texte qui stocke
            ` l'image de texte de l'enregistrement
        $vtDonnéesEnregistrement:=$vtDonnéesEnregistrement+Nom du champ($1;
                                        $vlChamp)+ ":"+Caractere(9)+$vtDonnéesChamp+CR
            ` Note : La méthode CR retourne Caractere(13) sous Mac OS et
            ` Caractere(13)+Caractere(10) sous Windows
    Fin de boucle
        ` Mettre l'image de texte de l'enregistrement dans le Presse-papiers
    FIXER TEXTE DANS CONTENEUR($vtDonnéesEnregistrement)
        ` Nommez le fichier d'Album dans le Dossier temporaire
    $vaNomDoc:=Dossier temporaire+"Album"+Chaine(1+(Hasard%99))
        ` Supprimer le fichier d'Album s'il existe (il faut tester une erreur ici)
    SUPPRIMER DOCUMENT($vaNomDoc)
        ` Créez le fichier d'Album
    REGLER SERIE(10;$vaNomDoc)
        ` Envoyer l'enregistrement entier dans le Presse-papiers
    ENVOYER ENREGISTREMENT($vpTable->)
        ` Fermer le fichier d'Album
    REGLER SERIE(11)
        ` Charger le fichier d'Album dans un BLOB
    DOCUMENT VERS BLOB($vaNomDoc;$vxDonnéesEnregistrement)
        ` Nous n'avons plus besoin du fichier d'Album
    SUPPRIMER DOCUMENT($vaNomDoc)
        ` Ajouter l'image complète de l'enregistrement dans le Presse-papiers
        ` Note: nous utilisons le type de données "4Drc" de façon arbitraire
    AJOUTER DONNEES AU CONTENEUR("4Drc";$vxDonnéesEnregistrement)
        ` Le Presse-papiers contient :
        ` (1) Une image de texte de l'enregistrement (comme illustré dans les copies
        ` d'écran ci-dessous)
        ` (2) Une image entière de l'enregistrement (y compris les images, sous-tables et les
        ` champs de type BLOB)
Fin de si
```

Lors de la saisie d'un enregistrement, si vous appliquez la méthode écrire enregistrement dans Presse-papiers à la table, le Presse-papiers contiendra le texte de l'enregistrement et également l'image entière de l'enregistrement.

Vous pouvez coller cette image de l'enregistrement dans un autre enregistrement, à l'aide de la méthode lire enregistrement dans Presse-papiers, qui est la suivante :

```
    ` Méthode lire enregistrement dans Presse-papiers
    ` lire enregistrement dans Presse-papiers ( Numéro )
    ` lire enregistrement dans Presse-papiers ( Numéro de table )
C_ENTIER LONG($1;$vlChamp;$vlTypeChamp;$vlPosCR;$vlPosColon)
C_POINTEUR($vpTable;$vpChamp)
C_ALPHA(255;$vaNomDoc)
C_BLOB($vxDonnéeesPressePapiers)
C_TEXTE($vtDonnéeesPressePapiers;$vtDonnéesChamp)

    ` Obtenir un pointeur vers la table dont le numéro est passé en tant que paramètre
$vpTable:=Table($1)
    ` S'il y a un enregistrement courant pour cette table
Si ((Numero enregistrement($vpTable->)>=0) | (Nouvel enregistrement($vpTable->)))
   Au cas ou
            ` Est-ce que le Presse-papiers contient une image entière de l'enregistrement ?
      : (Tester conteneur("4Drc")>0)
            ` Si oui, extraire le contenu du Presse-papiers
         LIRE DONNEES CONTENEUR("4Drc";$vxDonnéesPressePapiers)
            ` Nommer le fichier d'Album dans le Dossier temporaire
         $vaNomDoc:=Dossier temporaire+"Album"+Chaine(1+(Hasard%99))
            ` Supprimer le fichier d'Album s'il existe (il faut tester l'erreur ici)
         SUPPRIMER DOCUMENT($vaNomDoc)
            ` Enregistrer le BLOB dans le fichier d'Album
         BLOB VERS DOCUMENT($vaNomDoc;$vxDonnéesPressePapiers)
            ` Ouvrir le fichier d'Album
         REGLER SERIE(10;$vaNomDoc)
            ` Recevoir l'enregistrement entier du fichier d'Album
         RECEVOIR ENREGISTREMENT($vpTable->)
            ` Fermer le fichier d'Album
         REGLER SERIE(11)
            ` Nous n'avons plus besoin du fichier d'Album
         SUPPRIMER DOCUMENT($vaNomDoc)
            ` Est-ce que le Presse-papiers contient du texte ?
```

```
: (Tester conteneur("TEXT")>0)
      ` Extraire le texte du Presse-papiers
   $vtDonnéesPressePapiers:=Lire texte dans conteneur
      ` Initialiser le numéro de champ à incrémenter
   $vlChamp:=0
   Repeter
         ` Chercher la ligne de champ suivante dans le texte
      $vlPosCR:=Position(CR ;$vtDonnéesPressePapiers)
      Si ($vlPosCR>0)
            ` Extraire la ligne de champ
         $vtDonnéesChamp:=Sous chaine($vtDonnéesPressePapiers;1;
                                                      $vlPosCR-1)
            ` S'il y a un signe deux points ":"
         $vlPosColon:=Position(":";$vtDonnéesChamp)
         Si ($vlPosColon>0)
               ` Récupérer seulement les données de champ (supprimer le
               ` nom du champ)
            $vtDonnéesChamp:=Sous chaine($vtDonnéesChamp;
                                                   $vlPosColon+2)
         Fin de si
            ` Incrémenter le numéro du champ
         $vlChamp:=$vlChamp+1
            ` Le Presse-papiers peut contenir plus de données dont nous
            ` n'avons pas besoin...
         Si ($vlChamp<=Lire numero dernier champ($vpTable))
               ` Obtenir le type du champ
            LIRE PROPRIETES CHAMP($1;$vlChamp;$vlTypeChamp)
               ` Obtenir un pointeur vers le champ
            $vpChamp:=Champ($1;$vlChamp)
               ` Selon le type du champ, copier (ou non) le texte d'une
               ` manière appropriée
            Au cas ou
               : (($vlTypeChamp=Champ alphanumérique ) |
                                                ($vlTypeChamp=Texte ))
                  $vpChamp->:=$vtDonnéesChamp
               : (($vlTypeChamp=Numérique ) | ($vlTypeChamp=Entier ) |
                                          ($vlTypeChamp=Entier long ))
                  $vpChamp->:=Num($vtDonnéesChamp)
               : ($vlTypeChamp=Champ ou variable date )
                  $vpChamp->:=Date($vtDonnéesChamp)
```

```
                    : ($vlTypeChamp=Champ ou variable heure )
                        $vpChamp->:=Heure($vtDonnéesChamp)
                    : ($vlTypeChamp=Booléen )
                        $vpChamp->:=($vtDonnéesChamp="Oui")
                Sinon
                    ` Passer et ignorer les autres types de données
                Fin de cas
            Sinon
                    ` Tous les champs ont été affectés, sortir de la boucle
                $vtDonnéesPressePapiers:=""
            Fin de si
                ` Eliminer le texte qui vient d'être extrait
            $vtDonnéeesPressePapiers:=Sous chaine($vtDonnéeesPressePapiers;
                                                  $vlPosCR+Longueur(CR ))
        Sinon
                ` Aucun délimiteur trouvé, sortir de la boucle
            $vtDonnéesPressePapiers:=""
        Fin de si
            ` Répéter jusqu'à ce que nous ayons des données
    Jusque (Longueur($vtDonnéesPressePapiers)=0)
Sinon
    ALERTE("Le Presse-papiers ne contient pas de données pouvant être collées en
                                                  tant qu'enregistrement.")
    Fin de cas
Fin de si
```

Référence

EFFACER CONTENEUR, FIXER IMAGE DANS CONTENEUR, FIXER TEXTE DANS CONTENEUR.

Variables système

Si les données dans le BLOB sont correctement ajoutées au conteneur, la variable système OK prend la valeur 1. Sinon, OK est mise à 0 et une erreur peut être générée.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour ajouter les données du BLOB dans le conteneur, une erreur -108 est générée.

EFFACER CONTENEUR

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

EFFACER CONTENEUR efface entièrement le conteneur de données. Si le conteneur contient plusieurs instances des mêmes données, toutes les instances sont effacées. Après un appel à EFFACER CONTENEUR, le conteneur de données est vide.

Vous devez appeler EFFACER CONTENEUR une fois avant de placer des nouvelles données dans le conteneur à l'aide de la commande AJOUTER DONNEES AU CONTENEUR, car cette dernière n'efface pas le conteneur avant d'y coller des données.

Si vous appelez EFFACER CONTENEUR une fois et puis appelez AJOUTER DONNEES AU CONTENEUR plusieurs fois, vous pouvez couper ou copier les mêmes données sous des formats différents.

En revanche, les commandes ECRIRE TEXTE DANS CONTENEUR et ECRIRE IMAGE DANS CONTENEUR effacent automatiquement le conteneur avant d'y placer des données.

Exemple

(1) Le code suivant efface le conteneur puis y ajoute des données :

```
EFFACER CONTENEUR  ` Effacer le conteneur
AJOUTER DONNEES AU CONTENEUR("com.4d.private.picture.gif";$vxSomeData)
    ` Ajouter des images gif
AJOUTER DONNEES AU CONTENEUR("com.4d.private.text.rtf";$vxSylkData)
    ` Ajouter du texte RTF
```

(2) Reportez-vous à l'exemple de la commande AJOUTER DONNEES AU CONTENEUR.

Référence

AJOUTER DONNEES AU CONTENEUR.

FIXER FICHIER DANS CONTENEUR (cheminFichier)

Paramètre	Type		Description
cheminFichier	Chaîne	→	Chemin d'accès complet de fichier

Description

La commande FIXER FICHIER DANS CONTENEUR ajoute dans le conteneur de données le chemin d'accès complet passé dans le paramètre cheminFichier.

Cette commande permet de mettre en place des interfaces autorisant le glisser-déposer d'objets 4D vers des fichiers sur le bureau par exemple.

Note : Le conteneur de données est en lecture seule pendant l'événement formulaire Sur glisser. Il n'est pas possible d'utiliser cette commande dans ce contexte.

Référence

Lire fichier dans conteneur.

FIXER IMAGE DANS CONTENEUR (image)

Paramètre	Type		Description
image	Image	→	Image à placer dans le conteneur de données

Description

FIXER IMAGE DANS CONTENEUR place dans le conteneur de données une copie de l'image que vous avez passée dans image. Les données éventuellement présentes dans le conteneur sont préalablement effacées.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

L'image est transportée dans son format natif (jpeg, tif, png, etc.).

Après avoir placé l'image dans le conteneur, vous pouvez la récupérer à l'aide de la commande LIRE IMAGE DANS CONTENEUR ou par exemple LIRE DONNEES CONTENEUR("com.4d.private.picture.gif";...).

Exemple

Dans une fenêtre flottante, vous affichez un formulaire contenant le tableau tabNomEmployés qui liste les noms des employés stockés dans la table [Employés]. Chaque fois que vous cliquez sur un nom, vous voulez copier la photographie de l'employé dans le Presse-papiers. Dans la méthode objet du tableau, vous écrivez :

```
Si (tabNomEmployés#0)
    CHERCHER ([Employés];[Employés]Nom=tabNomEmployés{tabNomEmployés})
    Si (Taille image ([Employés]Photo)>0)
        FIXER IMAGE DANS CONTENEUR ([Employés]Photo)
            ` Copier la photo de l'employée
    Sinon
        EFFACER CONTENEUR  ` Aucune photo trouvée ou aucun enregistrement trouvé
    Fin de si
Fin de si
```

Référence

AJOUTER DONNEES AU CONTENEUR, LIRE IMAGE DANS CONTENEUR.

Variables et ensembles système

Si une copie de l'image est correctement collée dans le conteneur, la variable système OK prend la valeur 1. S'il n'y a pas assez de mémoire pour coller l'image dans le Presse-papiers, la variable système OK prend la valeur 0, mais aucune erreur n'est générée.

FIXER TEXTE DANS CONTENEUR (texte)

Paramètre	Type		Description
texte	Chaîne	→	Texte à placer dans le conteneur de données

Description

FIXER TEXTE DANS CONTENEUR place une copie du texte que vous avez passé dans texte dans le conteneur de données. Les données éventuellement présentes dans le conteneur sont auparavant effacées.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Vous pouvez récupérer le texte collé dans le conteneur de données à l'aide de la fonction Lire texte dans conteneur ou en appelant par exemple LIRE DONNEES CONTENEUR ("com.4d.private.text.native";...).

Les expressions de type Texte de 4D peuvent contenir jusqu'à 2 Go de texte.

Note : Le conteneur de données est en lecture seule pendant l'événement formulaire Sur glisser. Il n'est pas possible d'utiliser cette commande dans ce contexte.

Exemple

Référez-vous à l'exemple de la commande AJOUTER DONNEES AU CONTENEUR.

Référence

AJOUTER DONNEES AU CONTENEUR, Lire texte dans conteneur.

Variables et ensembles système

Si la copie du texte est correctement placée dans le conteneur de données, la variable système OK prend la valeur 1. S'il n'y a pas assez de mémoire pour placer une copie du texte dans le conteneur, la variable système OK prend la valeur 0, mais aucune erreur n'est générée.

LIRE DONNEES CONTENEUR (typeDonnées; données)

Paramètre	Type		Description
typeDonnées	Alpha	→	Type de données à extraire du conteneur
données	BLOB	←	Données extraites du conteneur

Description

LIRE DONNEES CONTENEUR retourne dans le champ ou la variable de type BLOB données les données qui se trouvent dans le conteneur de données et dont le type est passé dans typeDonnées.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Passez dans typeDonnées une valeur définissant le type de données à extraire. Vous pouvez passer une signature 4D, un type UTI (Mac OS), un nom/numéro de format (Windows), ou un type de 4 caractères (compatibilité). Pour plus d'informations sur ces types, reportez-vous à la section Gestion du conteneur de données.

Note : Il n'est pas possible de lire les données de type fichier avec cette commande, pour cela vous devez utiliser la commande Lire fichier dans conteneur.

Exemple

Les méthodes objet suivantes sont celles de deux boutons qui copient et collent des données dans le tableau taOptions (pop up menu, liste déroulante...) se trouvant dans le formulaire :

```
    ` Méthode objet bCopiertaOptions
Si (Taille tableau(taOptions)>0)  ` Est-ce qu'il y a quelque chose à copier ?
    VARIABLE VERS BLOB (taOptions;$vxClipData)
        ` Mettre les éléments du tableau dans un BLOB
    EFFACER CONTENEUR  ` Vider le Presse-papiers
    AJOUTER DONNEES AU CONTENEUR ("artx";taOptions)
        ` Le type de données est choisi arbitrairement
Fin de si
```

```
    ` Méthode objet bCollertaOptions
Si (Tester conteneur ("artx")>0)
        ` Est-ce qu'il y a les données du type "artx" dans le Presse-papiers?
    LIRE DONNEES CONTENEUR ("artx";$vxClipData)
        ` Extraire les données du Presse-papiers
    BLOB VERS VARIABLE ($vxClipData;taOptions)
        ` Remplir le tableau avec les données venant du BLOB
    taOptions:=0  ` Réinitialiser l'élément sélectionné du tableau
Fin de si
```

Référence

AJOUTER DONNEES AU CONTENEUR, LIRE IMAGE DANS CONTENEUR, Lire texte dans conteneur.

Variables système

Si les données sont extraites correctement, la variable OK prend la valeur 1. Sinon, elle prend la valeur 0.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour extraire les données, l'erreur -108 est générée.

Lire fichier dans conteneur (indiceN) → Chaîne

Paramètre	Type		Description
indiceN	Numérique	→	Nième fichier inclus dans le glisser
Résultat	Chaîne	←	Chemin d'accès de fichier extrait du conteneur de données

Description

La commande Lire fichier dans conteneur retourne le chemin d'accès absolu d'un fichier inclus dans une opération de glisser-déposer. Plusieurs fichiers pouvant être sélectionnés et déplacés simulanément, le paramètre indiceN permet de désigner un fichier parmi l'ensemble des fichiers sélectionnés.

S'il n'y a pas de Nième fichier dans le conteneur de données, la commande retourne une chaîne vide.

Exemple

L'exemple suivant permet de récupérer dans un tableau tous les chemins d'accès des fichiers inclus dans le glisser-déposer :

```
TABLEAU TEXTE($tabFichiers;0)
C_TEXTE($vtfichier)
C_ENTIER($n)
$n:=1
Repeter
    $vtfichier:=Lire fichier dans conteneur($n)
    Si($vtfichier#"")
        AJOUTER A TABLEAU($tabFichiers;$vtfichier)
        $n:=$n+1
    Fin de si
Jusque($vtfichier="")
```

Référence

FIXER FICHIER DANS CONTENEUR.

LIRE IMAGE DANS CONTENEUR (image)

Paramètre	Type		Description
image	Image	←	Image extraite du conteneur de données

Description

LIRE IMAGE DANS CONTENEUR retourne l'image présente dans le conteneur de données dans le champ ou la variable image.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

L'image est transportée dans son format natif (jpeg, tif, png, etc.). Dans le cas où la zone de destination accepte les formats natifs, l'image conserve son format, sinon elle est convertie au format PICT.

Exemple

Ci-dessous, la méthode objet d'un bouton affecte l'image au format jpeg ou gif présente dans le conteneur de données, s'il y en a une, au champ [Employés]Photo :

```
Si ((Tester conteneur ("com.4d.private.picture.jfif")>0) | (Tester conteneur
                                                ("com.4d.private.picture.gif")>0))
   LIRE IMAGE DANS CONTENEUR ([Employés]Photo)
Sinon
   ALERTE ("Le Presse-papiers ne contient pas d'image.")
Fin de si
```

Référence

LIRE DONNEES CONTENEUR, Lire texte dans conteneur, Tester conteneur.

Variables et ensembles système

Si l'image est correctement extraite, OK prend la valeur 1. Sinon, OK prend la valeur 0.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour extraire l'image, l'erreur -108 est générée.

Lire texte dans conteneur

Lire texte dans conteneur → Chaîne

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Chaîne	←	Texte présent dans le conteneur de données

Description

Lire texte dans conteneur retourne le texte présent dans le conteneur de données.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Si le conteneur de données contient du texte enrichi (par exemple au format RTF), le texte conserve ses attributs au moment du déposer ou du coller, si la zone de destination est compatible.

A noter que les champs et variables de type Texte de 4D peuvent contenir jusqu'à 2 Go de texte.

Référence

LIRE DONNEES CONTENEUR, LIRE IMAGE DANS CONTENEUR, Tester conteneur.

Variables et ensembles système

Si le texte est correctement extrait, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour extraire le texte, l'erreur -108 est générée.

LIRE TYPE DONNEES DANS CONTENEUR (signatures4D; typesNatifs{; nomsFormats})

Paramètre	Type		Description
signatures4D	Tab Texte	←	Signatures 4D des types de données
typesNatifs	Tab Texte	←	Types de données natifs
nomsFormats	Tab Texte	←	Noms des formats (Windows uniquement), chaînes vides sous Mac OS

Description

La commande LIRE TYPE DONNEES DANS CONTENEUR permet d'obtenir la liste des types de données présents dans le conteneur. Cette commande doit généralement être utilisée dans le contexte d'un glisser-déposer, dans le cadre des événements formulaire Sur déposer ou Sur glisser de l'objet de destination. Elle permet notamment de vérifier la présence d'un type de données spécifique dans le conteneur.

Cette commande retourne les types de données sous plusieurs formes différentes via deux (ou trois) tableaux :
• le tableau signatures4D contient les types de données exprimés à l'aide de leur signature 4D interne (par exemple "com.4d.private.picture.gif"). Si un type de données présent n'est pas reconnu par 4D, une chaîne vide ("") est retournée dans le tableau.
• le tableau typesNatifs contient les types de données exprimés à l'aide de leur type natif. Le format des types natifs diffère entre Mac OS et Windows :
- Sous Mac OS, les types natifs sont exprimés sous forme d'UTI (*UniformType Identifier*).
- Sous Windows, les types natifs sont exprimés sous forme de numéros, chaque numéro étant associé à un nom de format. Le tableau typesNatifs contient ces numéros sous forme de chaîne ("3", "12", etc.). Si vous souhaitez utiliser des libellés plus explicites, il est recommandé d'utiliser le tableau facultatif nomsFormats, qui contient le nom de format des types natifs sous Windows.
Le tableau typesNatifs permet de prendre en charge tout type de données présent dans le conteneur, y compris des données dont le type n'est pas référencé par 4D.
• Sous Windows, vous pouvez également passer le tableau nomsFormats, qui reçoit les noms des types de données présents dans le conteneur. Les valeurs retournées dans ce tableau peuvent être utilisées par exemple pour construire un pop up menu de sélection de format. Sous Mac OS, le tableau nomsFormats retourne des chaînes vides.

Pour plus d'informations sur les types de données pris en charge, reportez-vous à la section Gestion du conteneur de données.

Référence

Gestion du conteneur de données.

Tester conteneur

Tester conteneur (typeDonnées) → Numérique

Paramètre	Type		Description
typeDonnées	Chaîne	→	Type de données
Résultat	Numérique	←	Taille (en octets) des données présentes dans le conteneur ou code d'erreur

Description

Tester conteneur vous permet de savoir s'il y a des données du type typeDonnées dans le conteneur de données.

Note : Dans le cadre d'une opération de copier/coller, le conteneur de données correspond au Presse-papiers.

Si le conteneur de données est vide ou ne contient pas de données du type spécifié, la fonction retourne une erreur -102. Si le conteneur contient des données du type spécifié, la fonction retourne la taille des données exprimée en octets.

Passez dans typeDonnées une valeur définissant le type de données à tester. Vous pouvez passer une signature 4D, un type UTI (Mac OS), un nom/numéro de format (Windows), ou un type de 4 caractères (compatibilité). Pour plus d'informations sur ces types, reportez-vous à la section Gestion du conteneur de données.

Après avoir vérifié que le conteneur contient bien des données du type que vous voulez, vous pouvez les récupérer à l'aide d'une des commandes suivantes :

• Si le conteneur contient du texte, vous pouvez l'extraire à l'aide de la commande Lire texte dans conteneur, qui retourne une valeur texte. Sinon, vous pouvez utiliser la commande LIRE DONNEES CONTENEUR, qui retourne le texte dans un BLOB.

• Si le conteneur contient une image, vous pouvez l'extraire à l'aide de la commande LIRE IMAGE DANS CONTENEUR, qui retourne l'image dans un champ ou une variable. Sinon, vous pouvez utiliser la commande LIRE DONNEES CONTENEUR, qui retourne l'image dans un BLOB.

• Si le conteneur contient un chemin d'accès de fichier, vous pouvez l'extraire à l'aide de la commande Lire fichier dans conteneur, qui retourne le chemin d'accès du fichier.

• Pour tout type de données, vous pouvez utiliser la commande LIRE DONNEES CONTENEUR, qui retourne les données dans un BLOB.

Exemples

(1) L'exemple suivant teste si le Presse-papiers contient une image jpeg et, si oui, la copie dans une variable 4D :

```
Si (Tester conteneur ("com.4d.private.picture.jfif") > 0)
        ` Y a-t-il une image jpeg dans le Presse-papiers ?
    LIRE IMAGE DANS CONTENEUR ($vPicVariable)
        ` Si oui, extraire l'image du Presse-papiers
Sinon
    ALERTE("Il n'y a pas d'image dans le Presse-papiers.")
Fin de si
```

(2) Généralement, après un couper ou un copier, les applications placent des données de type Texte ou Image dans le Presse-papiers, ces deux types de données standard sont reconnus par la plupart des applications. Cependant, une application peut placer dans le Presse-papiers plusieurs copies des mêmes données sous des formats différents. Par exemple, chaque fois que vous copiez ou coupez un tableau, l'application tableur peut placer les données dans un format propriétaire — par exemple, 'SPSH' — ou dans les formats SYLK et TEXT. La copie 'SPSH' contient les données structurées dans le format interne de l'application. La copie SYLK contient les mêmes données, mais dans le format SYLK, reconnu par la plupart des tableurs. Enfin, la copie TEXT contient les mêmes données, mais sans les informations de formatage supplémentaires présentes dans les formats SYLK ou 'SPSH'. Donc, lorsque vous écrivez des routines de Couper/Copier/Coller entre 4D et une application tableur, en prenant l'hypothèse que vous connaissez la description du format 'SPSH' et que vous pouvez analyser les données SYLK, vous pouvez écrire le code suivant :

```
Au cas ou
        ` D'abord, vérifier si le Presse-papiers contient les données venant du tableur
    : (Tester conteneur ('SPSH') > 0)
        ` ...
        ` Ensuite, vérifier si le Presse-papiers contient des données au format SYLK
    : (Tester conteneur('SYLK') > 0)
        ` ...
        ` Enfin, vérifier si le Presse-papiers contient des données au format TEXT
    : (Tester conteneur ('TEXT') > 0)
        ` ...
Fin de cas
```

Autrement dit, vous essayez d'extraire du Presse-papiers la copie des données la plus riche en informations originales.

(3) Référez-vous à l'exemple de la commande AJOUTER DONNEES AU CONTENEUR.

Référence

LIRE DONNEES CONTENEUR, LIRE IMAGE DANS CONTENEUR, Lire texte dans conteneur.

9

Dates et heures

Ajouter a date

Ajouter a date (date; années; mois; jours) → Date

Paramètre	Type		Description
date	Date	→	Date à laquelle ajouter jours, mois et années
années	Numérique	→	Nombre d'années à ajouter à la date
mois	Numérique	→	Nombre de mois à ajouter à la date
jours	Numérique	→	Nombre de jours à ajouter à la date
Résultat	Date	←	Date résultante

Description

Ajouter a date ajoute années, mois et jours à la date que vous avez passée dans date, et retourne la date résultante.

Alors que les Opérateurs sur les dates vous permettent d'ajouter des jours à une date, Ajouter a date vous permet d'ajouter rapidement des mois et des années sans vous soucier du nombre de jours par mois ou des années bissextiles (comme vous devriez le faire avec l'opérateur "+" sur les dates).

Exemples

```
   ` Cette ligne calcule la date dans un an, le même jour
$vdDansUnAn:=Ajouter a date(Date du jour;1;0;0)

   ` Cette ligne calcule la date le mois prochain, le même jour
$vdMoisProchain:=Ajouter a date(Date du jour;0;1;0)

   ` Cette ligne fait la même chose que $vdDemain:=Date du jour+1
$vdDemain:=Ajouter a date(Date du jour;0;0;1)
```

Référence

Opérateurs sur les dates.

Annee de (date) → Numérique

Paramètre	Type		Description
date	Date	→	Date dont vous voulez extraire l'année
Résultat	Numérique	←	Nombre indiquant l'année de date

Description

Annee de retourne un nombre indiquant l'année de date.

Exemples

(1) L'exemple suivant illustre l'utilisation de Annee de. Les résultats sont assignés à la variable Résultat :

```
Résultat := Annee de (!25/12/96!)  ` Résultat prend la valeur 1996
Résultat := Annee de (!25/12/1996!)  ` Résultat prend la valeur 1996
Résultat := Annee de (!25/12/1896!)  ` Résultat prend la valeur 1896
Résultat := Annee de (!25/12/2096!)  ` Résultat prend la valeur 2096
Résultat := Annee de (Date du jour)
    ` Résultat prend comme valeur l'année de la date d'aujourd'hui
```

(2) Reportez-vous à l'exemple de la fonction Date du jour.

Référence

Jour de, Mois de.

Chaine heure (secondes) → Alpha

Paramètre	Type		Description
secondes	Numérique	→	Secondes écoulées depuis minuit
Résultat	Alpha	←	Heure sous forme de chaîne au format 24 heures

Description

La fonction Chaine heure retourne sous forme de chaîne alphanumérique sur 24 heures l'expression de type Heure passée dans secondes.

Le format appliqué à la chaîne est HH:MM:SS.

Si vous passez un nombre de secondes supérieur à celui qu'il y a dans un jour (86 400), Chaine heure continue d'ajouter les heures, les minutes et les secondes. Par exemple, Chaine heure (86401) retourne 24:00:01.

Note : Si vous voulez obtenir sous forme de chaîne une expression de type Heure dans des formats plus variés, utilisez la fonction Chaine.

Exemple

L'exemple suivant affiche une boîte de dialogue d'alerte avec le message “46800 secondes représentent 13:00:00” :

```
ALERTE ("46800 secondes représentent " + Chaine heure (46800))
```

Référence

Chaine, Heure.

Date (chaîneDate) → Date

Paramètre	Type		Description
chaîneDate	Alpha	→	Chaîne contenant la date à retourner
Résultat	Date	←	chaîneDate sous forme de Date

Description

La fonction Date extrait et retourne la date de la chaîne chaîneDate.

Le paramètre chaîneDate doit respecter les conventions d'écriture standard pour les formats de date. Dans une version française de 4D, la date doit être de la forme JJ/MM/AA (jour, mois, année). Le jour et le mois peuvent être composés d'un ou deux chiffres. L'année peut être composée de deux ou quatre chiffres. Si l'année comporte deux chiffres, Date considère que la date appartient au XXe ou au XXIe siècle en fonction de la valeur saisie. Par défaut, la valeur pivot est 30 :
• si la valeur saisie est supérieure ou égale à 30, 4D considère que la date appartient au XXe siècle et ajoute 19 devant la valeur.
• si la valeur saisie est inférieure à 30, 4D considère que la date appartient au XXIe siècle et ajoute 20 devant la valeur.
Ce mécanisme peut être modifié à l'aide de la commande SIECLE PAR DEFAUT.
Les caractères de séparation de date autorisés sont les suivants : barre oblique (/), espace, point (.), virgule (,) et tiret (-).
Date ne vérifie pas la validité de la date passée dans chaîneDate. Si une date erronée (telle que "13/35/94") est passée, Date retourne une date invalide. Si chaîneDate ne peut être interprétée comme une date (par exemple, "aa/12/94"), une date nulle (!00/00/00!) est retournée. Il est de votre ressort de tester la validité de chaîneDate.

Exemples

(1) L'exemple suivant demande à l'utilisateur de saisir une date. La chaîne saisie est convertie en date et stockée dans la variable DemDate :

```
DemDate := Date (Demander ("Saisissez une date :"; Chaine (Date du jour)))
Si (OK=1)
   ` Faire quelque chose avec la date
                        Fin de si
```

(2) L'exemple suivant retourne la chaîne "12/12/97" sous forme de date :

```
vDate:=Date("12/12/97")
```

Référence
SIECLE PAR DEFAUT.

Date du jour {(*)} → Date

Paramètre	Type		Description
*		→	Retourne la date du jour du serveur
Résultat	Date	←	Date du jour

Description

Date du jour retourne la date courante telle que définie dans l'horloge système de la machine.

4D Server : Si vous passez le paramètre astérisque (*) — lors d'une exécution sur un poste 4D Client —, la fonction retourne la date du jour telle que définie dans l'horloge du poste serveur.

Exemples

(1) L'exemple suivant fait apparaître une boîte de dialogue d'alerte affichant la date du jour :

```
ALERTE("Nous sommes le " + Chaine(Date du jour) + ".")
```

(2) Vous développez une application pour le marché international. Vous souhaitez savoir si la version de 4D avec laquelle votre application est exécutée fonctionne avec des dates formatées en MM/JJ/AAAA (version US) ou JJ/MM/AAAA (version française). Cette information est nécessaire pour vous permettre, par exemple, de personnaliser correctement les zones de saisie.

La méthode projet suivante vous permet de traiter cette question :

```
` Méthode projet (fonction) Format date système
` Format date système -> Chaine
` Format date système -> Format de données 4D par défaut

C_ALPHA(31;$0;$vsDate;$vsMJA;$vsMois;$vsJour;$vsAnnée)
C_ENTIER LONG($1;$vlPos)
C_DATE($vdDate)
```

```
    ` Récupérer une date dans laquelle les valeurs de mois, de jour et d'année
    `sont toutes différentes
$vdDate:=Date du jour
Repeter
    $vsMois:=Chaine(Mois de($vdDate))
    $vsJour:=Chaine(Jour de($vdDate))
    $vsAnnée:=Chaine(Annee de($vdDate)%100)
    Si (($vsMois=$vsJour) | ($vsMois=$vsAnnée) | ($vsJour=$vsAnnée))
        OK:=0
        $vdDate:=$vdDate+1
    Sinon
        OK:=1
    Fin de si
Jusque (OK=1)
$0:="" ` Initialisation du résultat de la fonction
$vsDate:=Chaine($vdDate)
$vlPos:=Position("/";$vsDate) ` Trouver le premier séparateur / dans la chaîne ../../..
$vsMJA:=Sous chaine($vsDate;1;$vlPos-1) ` Extraire les premiers chiffres de la date
    ` Eliminer les premiers chiffres et le premier séparateur /
$vsDate:=Sous chaine($vsDate;$vlPos+1)
Au cas ou
    : ($vsMJA=$vsMois) ` Les chiffres expriment le mois
        $0:="MM"
    : ($vsMJA=$vsJour) ` Les chiffres expriment le jour
        $0:="JJ"
    : ($vsMJA=$vsAnnée) ` Les chiffres expriment l'année
        $0:="AAAA"
Fin de cas
$0:=$0+"/" ` Commencer à construire le résultat de la fonction
$vlPos:=Position("/";$vsDate) ` Trouver le deuxième séparateur dans la chaîne ../..
$vsMJA:=Sous chaine($vsDate;1;$vlPos-1) ` Extraire les chiffres suivants de la date
$vsDate:=Sous chaine($vsDate;$vlPos+1) ` Réduire la chaîne aux derniers chiffres de la date
Au cas ou
    : ($vsMJA=$vsMois) ` Les chiffres expriment le mois
        $0:=$0+"MM"
    : ($vsMJA=$vsJour) ` Les chiffres expriment le jour
        $0:=$0+"JJ"
    : ($vsMJA=$vsAnnée) ` Les chiffres expriment l'année
        $0:=$0+"AAAA"
Fin de cas
```

```
$0:=$0+"/"  ` Poursuivre la construction du résultat de la fonction
Au cas ou
    : ($vsDate=$vsMois)  ` Les chiffres expriment le mois
        $0:=$0+"MM"
    : ($vsDate=$vsJour)  ` Les chiffres expriment le jour
        $0:=$0+"DD"
    : ($vsDate=$vsAnnée)  ` Les chiffres expriment l'année
        $0:=$0+"AAAA"
Fin de cas
    ` A ce moment, $0 vaut soit MM/JJ/AAAA soit JJ/MM/AAAA, ou encore...
```

Référence

Annee de, Jour de, Mois de, Opérateurs sur les dates.

Heure (chaineHeure) → Heure

Paramètre	Type		Description
chaineHeure	Alpha	→	Chaîne à retourner sous forme d'heure
Résultat	Heure	←	Heure définie par chaineHeure

Description

La fonction Heure retourne, sous la forme d'une expression de type Heure, l'heure définie dans la chaîne chaineHeure.

Le paramètre chaineHeure doit contenir une heure exprimée dans l'un des formats d'heure standard de 4D correspondant à la langue de votre système (pour plus d'informations, reportez-vous à la description de la commande Chaine).

Exemple

L'exemple suivant affiche une boîte de dialogue d'alerte avec le message “1:00 P.M. = 13 heures 0 minute.” :

```
ALERTE ("1:00 P.M. = " + Chaine (Heure ("13:00:00"); Heure Minute))
```

Référence

Chaine, Chaine heure.

Heure courante {(*)} → Heure

Paramètre	Type		Description
*		→	Retourne l'heure courante sur le poste serveur
Résultat	Heure	←	Heure courante

Description

La fonction Heure courante retourne l'heure courante définie dans l'horloge de votre système.

L'heure courante est toujours comprise entre 00:00:00 et 23:59:59. Vous pouvez utiliser les fonctions Chaine ou Chaine heure pour convertir en chaîne alphanumérique l'expression de type heure retournée par Heure courante.

4D Server : Si vous passez le paramètre astérisque (*) — lors d'une exécution sur un poste 4D Client —, la fonction retourne l'heure courante telle que définie dans l'horloge du poste serveur.

Exemples

(1) L'exemple suivant vous permet de mesurer la durée d'une opération. Dans cet exemple, vous voulez chronométrer la méthode longueOpération :

```
CelaPrend := Heure courante  ` Stocker l'heure de départ
longueOpération  ` Effectuer l'opération
ALERTE ("L'opération a pris " + Chaine (Heure courante – CelaPrend))  ` Afficher la durée
```

(2) L'exemple suivant extrait les heures, minutes et secondes de l'heure courante :

```
$vhMaintenant:= Heure courante
ALERTE("L'heure courante est : "+Chaine($vhMaintenant\3600))
ALERTE("La minute courante est : "+Chaine(($vhMaintenant\60)%60))
ALERTE("La seconde courante est : "+Chaine($vhMaintenant%60))
```

Référence

Chaine, Nombre de millisecondes, Nombre de ticks, Opérateurs sur les heures.

Jour de (date) → Numérique

Paramètre	Type		Description
date	Date	→	Date dont vous voulez extraire le jour
Résultat	Numérique	←	Jour du mois de date

Description

Jour de retourne le jour du mois de date.

Exemples

(1) L'exemple suivant illustre l'utilisation de Jour de. Les valeurs retournées sont stockées dans la variable Résultat. Les commentaires décrivent la valeur de Résultat :

```
Résultat := Jour de (!25/12/96!) ` Résultat vaut 25
Résultat := Jour de (Date du jour) ` Résultat prend la valeur du jour d'aujourd'hui
```

(2) Reportez-vous à l'exemple de la fonction Date du jour.

Référence

Annee de, Mois de, Numero du jour.

Mois de (laDate) → Numérique

Paramètre	Type		Description
laDate	Date	→	Date dont vous voulez extraire le mois
Résultat	Numérique	←	Nombre indiquant le mois de date

Description

Mois de retourne un nombre représentant le numéro du mois de laDate.

Note : C'est le numéro du mois est retourné, et non son nom (reportez-vous à l'exemple ci-dessous).

Pour comparer la valeur retournée par cette fonction, 4D fournit les constantes prédéfinies suivantes, placées dans le thème "Jours et mois" :

Constantes	Type	Valeur
Janvier	Entier long	1
Février	Entier long	2
Mars	Entier long	3
Avril	Entier long	4
Mai	Entier long	5
Juin	Entier long	6
Juillet	Entier long	7
Août	Entier long	8
Septembre	Entier long	9
Octobre	Entier long	10
Novembre	Entier long	11
Décembre	Entier long	12

Exemples

(1) L'exemple suivant illustre l'utilisation de Mois de. Les valeurs retournées sont assignées à la variable Résultat. Les commentaires fournissent les valeurs de Résultat :

```
Résultat := Mois de (!25/12/96!) ` Résultat vaut 12
Résultat := Mois de (Date du jour) ` Résultat prend la valeur du mois d'aujourd'hui
```

(2) Reportez-vous à l'exemple de la fonction Date du jour.

Référence

Annee de, Jour de.

Nombre de ticks

Nombre de ticks → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Nombre de ticks (60ème de seconde) écoulés depuis le démarrage de la machine

Description

Nombre de ticks retourne le nombre de ticks (1 tick = 1/60ème de seconde) écoulés depuis le démarrage de la machine.

Note : Nombre de ticks retourne une valeur de type Entier long.

Exemple

Référez-vous à l'exemple de la fonction Nombre de millisecondes.

Référence

Heure courante, Nombre de millisecondes.

Nombre de millisecondes

version 6.0

Nombre de millisecondes → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Nombre de millisecondes (1000ème de seconde) écoulées depuis le démarrage de la machine

Description

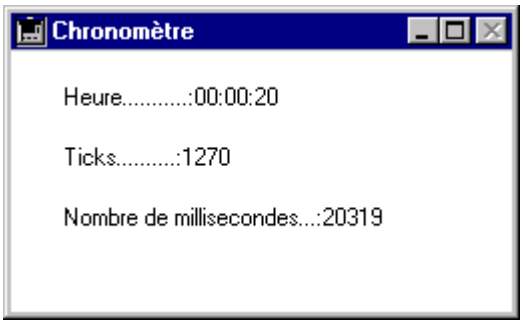
Nombre de millisecondes retourne le nombre de millisecondes (1 milliseconde = 1/1000ème de seconde) écoulées depuis le démarrage de la machine.

Exemple

Le code suivant :

```
Creer fenetre (100;100;350;230;0;"Chronomètre")
$vhDébutHeure:=Heure courante
$vlDébutTicks:=Nombre de ticks
$vrDébutMillisecondes:=Nombre de millisecondes
Repeter
   POSITION MESSAGE (2;1)
   MESSAGE ("Heure.....:"+Chaine (Heure courante -$vhDébutHeure))
   POSITION MESSAGE (2;3)
   MESSAGE ("Ticks.....:"+Chaine (Nombre de ticks -$vlDébutTicks))
   POSITION MESSAGE (2;5)
   MESSAGE ("Nombre de millisecondes...:"+Chaine (Nombre de millisecondes -
                                                  $vrDébutMillisecondes))
Jusque ((Heure courante -$vhDébutHeure)>=†00:01:00†)
FERMER FENETRE
```

... affiche la fenêtre suivante pendant une minute :



Référence

Heure courante, Nombre de ticks.

Numero du jour (laDate) → Numérique

Paramètre	Type		Description
laDate	Date	→	Date dont vous souhaitez connaître le numéro du jour
Résultat	Numérique	←	Numéro représentant le jour de la semaine auquel date correspond

Description

La fonction Numero du jour retourne un numéro représentant le jour de la semaine auquel laDate correspond.

Note : Si une date nulle est passée à Numero du jour, la fonction retourne 2.

4D fournit les constantes prédéfines suivantes, placées dans le thème "Jours et mois" :

Constantes	Type	Valeur
Lundi	Entier long	2
Mardi	Entier long	3
Mercredi	Entier long	4
Jeudi	Entier long	5
Vendredi	Entier long	6
Samedi	Entier long	7
Dimanche	Entier long	1

Note : Numero du jour retourne une valeur comprise entre 1 et 7. Pour obtenir le numéro du jour dans le sens "date du mois", utilisez la fonction Jour de.

Exemple

L'exemple suivant est une fonction qui retourne le jour d'aujourd'hui sous forme de chaîne :

```
$Jour := Numero du jour (Date du jour)
   ` $Jour prend comme valeur le numéro du jour courant
Au cas ou
   : ($Jour = 1)
                        $0 := "Dimanche"
```

```
        : ($Jour = 2)
        $0 := "Lundi"
        : ($Jour = 3)
        $0 := "Mardi"
        : ($Jour = 4)
        $0 := "Mercredi"
        : ($Jour = 5)
        $0 := "Jeudi"
        : ($Jour = 6)
        $0 := "Vendredi"
        : ($Jour = 7)
        $0 := "Samedi"
    Fin de cas
```

Référence

Jour de.

SIECLE PAR DEFAUT (siècle{; anPivot})

Paramètre	Type	Description
siècle	Numérique →	Siècle par défaut (moins un) lors de la saisie d'années sur 2 chiffres
anPivot	Numérique →	Année pivot lors de la saisie d'années sur 2 chiffres

Description

La commande SIECLE PAR DEFAUT vous permet de définir le siècle courant par défaut et l'année pivot adoptés par 4D lorsque des dates sont saisies avec seulement deux chiffres pour l'année.

L'année pivot indique la valeur au-dessous de laquelle une année saisie sur deux chiffres sera interprétée comme appartenant au siècle suivant :
• si l'année saisie est supérieure ou égale à l'année pivot, elle appartient au siècle courant,
• si l'année saisie est inférieure à l'année pivot, elle appartient au siècle suivant (relativement au siècle courant par défaut).

Par défaut, 4D présume que les dates appartiennent au 20e siècle et utilise la valeur 30 comme année pivot :
• Si vous saisissez la date 25/01/97, 4D considère que vous voulez indiquer le 25 janvier 1997
• Si vous saisissez la date 25/01/30, 4D considère que vous voulez indiquer le 25 janvier 1930
• Si vous saisissez la date 25/01/29, 4D considère que vous voulez indiquer le 25 janvier 2029
• Si vous saisissez la date 25/01/07, 4D considère que vous voulez indiquer le 25 janvier 2007

La commande SIECLE PAR DEFAUT permet de modifier ce comportement par défaut. Une fois exécutée, elle prend effet immédiatement.
Vous pouvez passer uniquement un siècle par défaut, ou un siècle par défaut et une année pivot.

Si vous passez uniquement un nouveau siècle par défaut moins un dans siècle, 4D interprétera toutes les années saisies sur deux chiffres comme appartenant à ce siècle.
Par exemple, après l'appel :

```
SIECLE PAR DEFAUT(20) ` Fixer le 21e siècle comme siècle par défaut
```

• Si vous saisissez la date 25/01/97, 4D considère que vous voulez indiquer le 25 janvier 2097
• Si vous saisissez la date 25/01/07, 4D considère que vous voulez indiquer le 25 janvier 2007

En outre, vous pouvez spécifier le paramètre anPivot.
Par exemple, après l'appel :

SIECLE PAR DEFAUT(19;95) ` Fixer le 21e siècle comme siècle par défaut si l'année est inférieure à 95
• Si vous saisissez la date 25/01/97, 4D considère que vous voulez indiquer le 25 janvier 1997
• Si vous saisissez la date 25/01/07, 4D considère que vous voulez indiquer le 25 janvier 2007

Notez que cette commande n'affecte que l'interprétation des dates lorsque les années sont saisies sur 2 chiffres. Quels que soient les paramètres passés à SIECLE PAR DEFAUT :
• Si vous saisissez la date 25/01/1997, 4D considère que vous voulez indiquer le 25 janvier 1997
• Si vous saisissez la date 25/01/2097, 4D considère que vous voulez indiquer le 25 janvier 2097
• Si vous saisissez la date 25/01/1907, 4D considère que vous voulez indiquer le 25 janvier 1907
• Si vous saisissez la date 25/01/2007, 4D considère que vous voulez indiquer le 25 janvier 2007

Cette commande affecte uniquement la saisie des données. Elle n'influe pas sur le stockage des données, les calculs, etc.

10

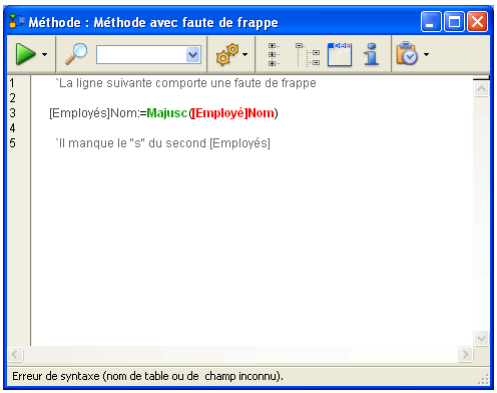
Débogueur

Lorsque vous développez et testez vos méthodes, il est important que vous puissiez repérer et corriger les erreurs qui peuvent s'y être glissées.

Vous pouvez commettre plusieurs types d'erreurs en utilisant le langage : des fautes de frappe, des erreurs de syntaxe ou liées à l'environnement, des erreurs logiques ou de conception, ou encore des erreurs d'exécution (Runtime).

Fautes de frappe

Les fautes de frappe sont détectées directement dans l'**éditeur de méthodes** et sont indiquées par un libellé de couleur rouge ainsi qu'un message dans la zone d'information située en bas de la fenêtre. Cette fenêtre signale une faute de frappe :



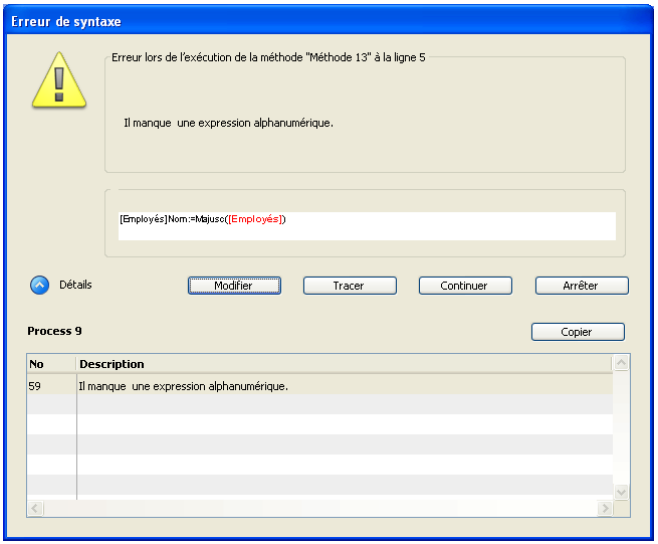
Note : Les commentaires ont été manuellement ajoutés pour cet exemple. Seule la couleur est modifiée par 4D à l'endroit de l'erreur.

De telles fautes de frappe provoquent généralement des erreurs de syntaxe (dans ce cas, le nom de la table est inconnu). La zone d'informations de la fenêtre affiche la cause de l'erreur au moment de la validation de la ligne d'instructions.

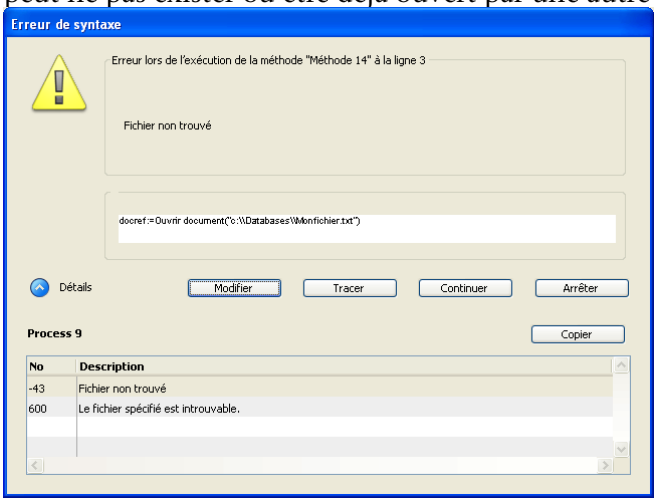
Vous pouvez alors corriger l'erreur de frappe et appuyer sur la touche **Entrée** (du clavier numérique) pour valider la correction. Pour plus d'informations sur l'éditeur de méthodes, reportez-vous au manuel *Mode Développement* de 4D.

Erreurs de syntaxe ou liées à l'environnement

Certaines erreurs ne peuvent être détectées que lorsque vous exécutez la méthode. La fenêtre Erreur de syntaxe est affichée lorsqu'une telle erreur est détectée. Par exemple :



Ici, l'erreur provient du fait que le nom d'une table est passé à la commande Majusc, qui attend une expression de type Texte. Dans cette image, la zone "Détail" est déployée afin d'afficher la dernière erreur et son numéro.
Parfois, il peut arriver que vous n'ayez pas assez de mémoire pour créer un tableau ou un BLOB. Ou bien, lorsque vous cherchez à accéder à un document sur votre disque, ce document peut ne pas exister ou être déjà ouvert par une autre application.

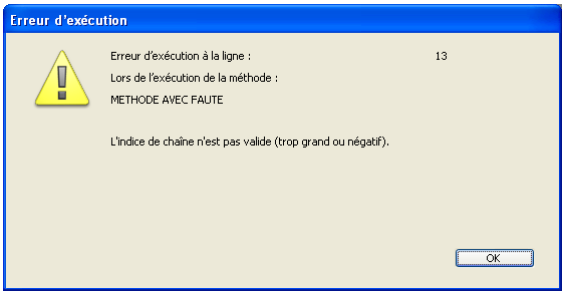


Ces erreurs ne se produisent pas à cause de votre code. Simplement, ce sont des choses qui

• Une méthode ne fait pas exactement ce que vous attendiez parce que vous ne testez pas la présence éventuelle d'un paramètre optionnel.

Erreurs d'exécution

En mode Application, vous pouvez obtenir des erreurs que vous n'avez jamais vues en mode interprété. Voici un exemple :



Ce message vous indique que vous essayez d'accéder à un caractère dont la position se trouve au-delà de la longueur de la chaîne. Pour trouver rapidement l'origine du problème, notez le nom de la méthode et le numéro de la ligne, ouvrez votre base en mode interprété puis allez à la méthode et à la ligne indiquées.

Que faire lorsque vous rencontrez une erreur ?

Les erreurs sont chose commune. Il est rare d'écrire une grande quantité de lignes de code sans générer d'erreur. Traiter et corriger les erreurs est tout aussi naturel.

Grâce à son environnement multitâche, 4D vous permet de modifier et d'exécuter rapidement vos méthodes : vous n'avez qu'à passer d'une fenêtre à une autre. De plus, vous découvrirez combien il est simple et rapide de repérer vos erreurs en utilisant le Débogueur.

Un réflexe courant chez les débutants lorsqu'une erreur est détectée est de cliquer sur le bouton **Arrêter** dans la fenêtre d'erreur de syntaxe, de retourner à l'éditeur de méthodes, et d'essayer de deviner ce qui se passe en regardant le code. Ne faites pas cela ! Vous gagnerez un temps considérable en utilisant **toujours** le Débogueur.

• S'il se produit une erreur de syntaxe inattendue, utilisez le Débogueur.
• S'il se produit une erreur d'environnement, utilisez le Débogueur.
• S'il se produit tout autre type d'erreur, utilisez le Débogueur.

Dans 99 % des cas, le Débogueur affiche l'information dont vous avez besoin pour comprendre la cause de l'erreur. Vous pouvez alors la corriger.

Astuce : Passez quelques heures à apprendre et expérimenter le Débogueur. Vous gagnerez des journées et des mois dans l'avenir.

La phase de développement représente une autre occasion d'utiliser le Débogueur. Imaginons que vous deviez écrire un algorithme plus complexe que d'habitude. Une fois le code écrit, vous ne pouvez être certain qu'il fonctionne tant que vous ne l'aurez pas testé. Au lieu de passer directement en exécution, vous pouvez d'abord le vérifier en insérant la commmande TRACE au début de votre code. Ensuite, exécutez le code au pas à pas pour contrôler ce qui se passe.

Conclusion générale
Utilisez le Débogueur.

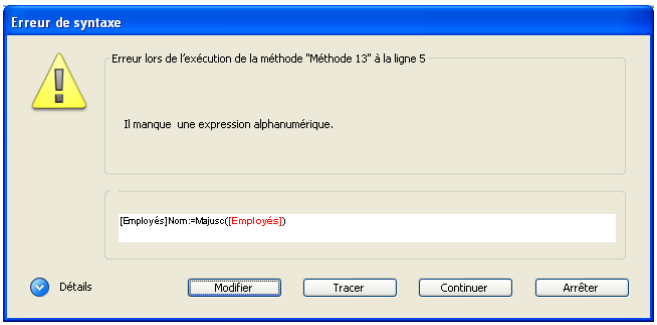
Référence

APPELER SUR ERREUR, Débogueur, Fenêtre d'erreur de syntaxe, Liste des points d’arrêt, Points d'arrêt sur commandes, Raccourcis du débogueur.

La Fenêtre d'erreur de syntaxe s'affiche lorsque l'exécution d'une méthode est interrompue. L'exécution de la méthode peut être interrompue pour l'une des deux raisons suivantes :

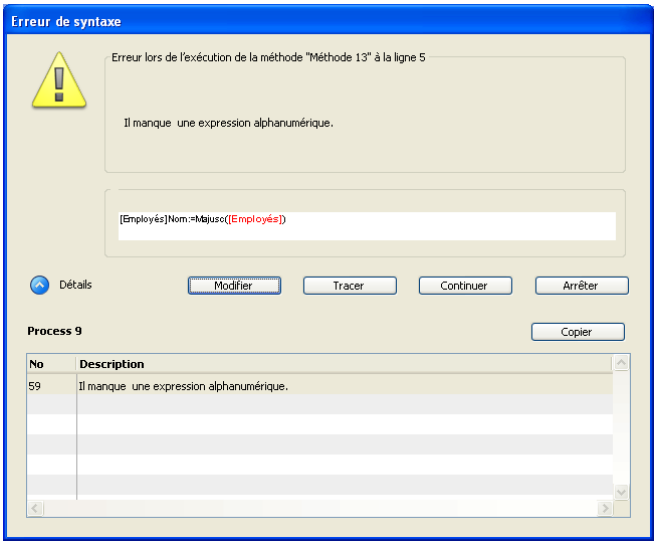
- 4D interrompt la méthode car une erreur l'empêche de poursuivre son exécution.
- Vous provoquez une interruption utilisateur en appuyant sur **Alt+Clic** (sous Windows) ou **Option+Clic** (sous Mac OS) pendant l'exécution de la méthode.

Voici une Fenêtre d'erreur de syntaxe :



Le texte situé dans la zone supérieure de la fenêtre affiche un message décrivant l'erreur. La partie inférieure fait apparaître la ligne exécutée au moment où l'erreur est survenue ; l'emplacement précis où est survenue l'erreur est sélectionné.

Le bouton **Détails** permet de déployer la partie inférieure de la fenêtre affichant la "pile" d'erreurs liées au process :



La fenêtre comporte cinq boutons : **Arrêter**, **Tracer**, **Continuer**, **Modifier** et (si la fenêtre est déployée) **Copier**.

• **Arrêter** : La méthode est interrompue et vous retournez à l'endroit où vous vous trouviez avant de commencer l'exécution de la méthode. Si une méthode formulaire ou une méthode objet s'exécutent en réponse à un événement, elles sont stoppées et vous retournez au formulaire. Si la méthode s'exécute à partir du mode Application, vous retournez dans ce mode.

• **Tracer** : Vous entrez dans le mode Trace et la fenêtre du Débogueur est affichée. Si la ligne courante a été partiellement exécutée, il se peut que vous soyez obligé de cliquer plusieurs fois sur le bouton **Tracer**. Lorsque la ligne est terminée, la fenêtre du Débogueur s'affiche.

• **Continuer** : L'exécution continue. La ligne contenant l'erreur peut avoir été partiellement exécutée — tout dépend de l'endroit où se trouvait l'erreur. Continuez avec prudence — l'erreur peut empêcher que le reste de la méthode s'exécute correctement. Généralement, il vaut mieux ne pas continuer. Vous pouvez cliquer sur Continuer si l'erreur se trouve dans un appel mineur, comme par exemple CHANGER TITRE FENETRE, qui n'empêche pas de continuer l'exécution et le test du code. Vous pouvez vous concentrer sur le code le plus important, et corriger les erreurs mineures ultérieurement.

• **Modifier** : L'exécution de la méthode est totalement interrompue. 4D passe en mode Développement. La méthode dans laquelle l'erreur est survenue est ouverte dans l'éditeur de méthodes, ce qui vous permet de la corriger. Utilisez cette option lorsque vous avez identifié immédiatement l'erreur et que vous pouvez la corriger sans qu'il soit nécessaire d'effectuer d'autres investigations.

• **Copier** : Ce bouton provoque la copie des informations de débogage dans le Presse-papiers. Ces informations décrivent l'environnement interne de l'erreur (numéro, composant interne, etc.). Elles sont formatées en texte tabulé. Une fois que vous avez cliqué sur ce bouton, vous pouvez coller le contenu du Presse-papiers dans un fichier texte, un tableur, un message email, etc. à des fins d'analyse.

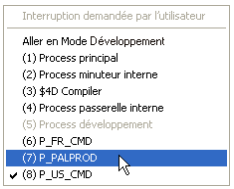
Référence

APPELER SUR ERREUR, Débogueur, Un débogueur, pour quoi faire ?.

Le mot Débogueur provient du terme anglais bug (insecte ou punaise) qui se "traduit" en français par bogue . Une bogue dans une méthode est une erreur que vous désirez éliminer. Lorsqu'une erreur se produit dans votre code, ou lorsque vous désirez contrôler l'exécution de vos méthodes, vous utilisez le débogueur. Un débogueur vous permet de trouver les bogues en exécutant pas à pas vos méthodes et en fournissant toutes les informations nécessaires. Ce procédé s'appelle le **mode Trace**.

Pour appeler la fenêtre du débogueur et afficher puis tracer les méthodes, vous pouvez :

• Cliquer sur le bouton **Tracer** dans la fenêtre d'erreur de syntaxe,
• Utiliser la commande TRACE,
• Cliquer sur le bouton **Déboguer** dans la fenêtre d'exécution de méthode.
• Utiliser la combinaison **Alt+Maj+clic droit** (sous Windows) ou **Control+Option+Commande+clic** (sous Mac OS) pendant l'exécution d'une méthode, puis choisir le process à tracer dans le pop up menu qui apparaît :

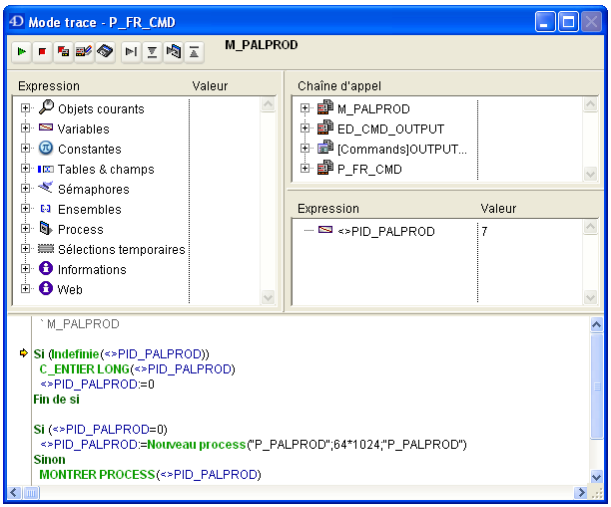


• Cliquer sur le bouton **Tracer** en mode Développement, lorsqu'un process est sélectionné dans la page **Process** de l'Explorateur d'exécution.
• Créer ou modifier un point d'arrêt dans la fenêtre d'édition d'une méthode, ou dans les pages **Point d'arrêt** et **Arrêt sur commande** de l'Explorateur d'exécution.

Note : Lorsque vous tracez un process en cours d'exécution, la fenêtre du débogueur s'affiche instantanément. Si vous tracez un process qui n'est pas en cours d'exécution (process endormi, en attente d'événement, etc...), le débogueur "enregistre" la requête et n'apparaîtra qu'au moment où le process aura repris l'exécution du code.

Note : Si votre base de données est dotée d'un système de mots de passe, seuls le Super_Utilisateur et les utilisateurs possédant les privilèges d'accès au développement peuvent tracer des méthodes.

Voici la fenêtre du débogueur :

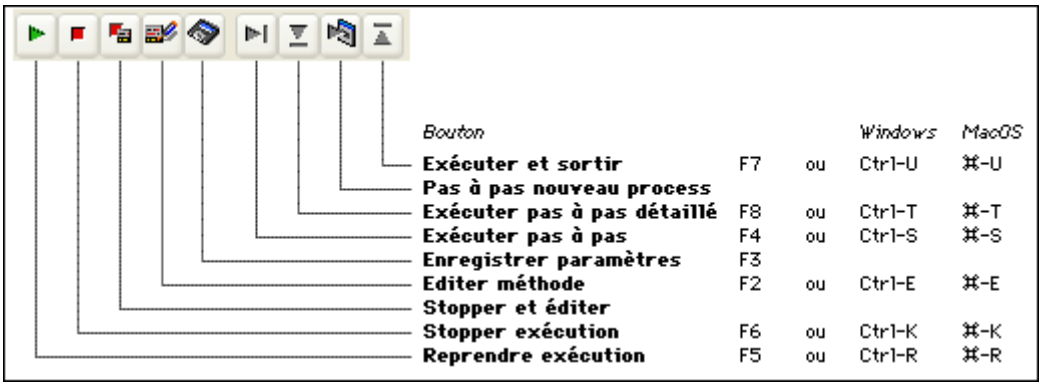


Vous pouvez déplacer la fenêtre du débogueur et/ou redimensionner toutes les zones internes de la fenêtre comme vous le souhaitez. L'affichage ultérieur d'une nouvelle fenêtre du débogueur reprend la configuration (taille et position de la fenêtre, emplacement des lignes de division et contenu de la zone d'évaluation des expressions) de la dernière fenêtre affichée dans la même session.

4D est un environnement multitâche. Dans le cas de plusieurs process s'exécutant simultanément, vous pouvez tracer chacun d'entre eux de manière indépendante. Chaque process peut avoir sa propre fenêtre de débogueur.

Barre d'outils de contrôle d'exécution

La **barre d'outils de contrôle d'exécution**, située en haut de la fenêtre du débogueur, comporte neuf boutons. Voici leur description ainsi que leurs raccourcis clavier associés :



Bouton 'Reprendre exécution'

Arrêt du mode Trace et reprise du cours normal de l'exécution de la méthode.

Note : La combinaison **Maj+F5** ou **Maj+clic** sur le bouton **Reprendre exécution** provoque la reprise de l'exécution avec désactivation de tous les appels à TRACE suivants dans le process courant.

Bouton 'Stopper exécution'

La méthode s'arrête et vous retournez là où vous étiez avant son exécution. Si vous étiez en train de tracer une méthode formulaire ou une méthode objet s'exécutant en réponse à un événement, elle s'arrête et vous retournez au formulaire. Si vous traciez une méthode s'exécutant à partir du mode Application, vous retournez à ce mode.

Bouton 'Stopper et éditer'

La méthode s'arrête comme lorsque vous cliquez sur **Stopper exécution**. De plus, 4D affiche dans la fenêtre de l'éditeur de méthodes la méthode qui s'exécutait au moment où vous avez cliqué sur le bouton.

Conseil : Utilisez ce bouton lorsque vous connaissez les modifications à apporter à votre code, et le moment où elles doivent être effectuées pour pouvoir poursuivre le test de vos méthodes. Une fois vos modifications effectuées, ré-exécutez la méthode.

Bouton 'Editer méthode'

Ce bouton se comporte comme le bouton **Stopper et éditer**, à la différence près qu'il n'annule pas l'exécution en cours. L'exécution de la méthode est simplement suspendue à l'instant du clic sur le bouton. 4D affiche dans la fenêtre de l'éditeur de méthodes la méthode qui s'exécutait au moment où vous avez cliqué sur le bouton **Editer méthode**.

Important : Vous pouvez modifier cette méthode, mais ces modifications n'apparaîtront pas, ou ne s'exécuteront pas dans l'instance de la méthode tracée dans la fenêtre du débogueur. Ce n'est qu'une fois que la méthode aura été stoppée ou entièrement exécutée que les modifications pourront apparaître. En d'autres termes, il faut recharger la méthode pour que les modifications soient prises en compte.

Conseil : Utilisez ce bouton lorsque vous connaissez les modifications à apporter à votre code et lorsqu'elles n'interfèrent pas avec le reste du code qui doit être exécuté ou tracé.

Astuce : Les méthodes objet sont rechargées pour chaque événement. Si vous tracez une méthode objet (par exemple en réponse à un clic sur un bouton) vous n'avez pas besoin de quitter le formulaire. Vous pouvez modifier la méthode objet, sauvegarder les modifications, puis retourner au formulaire et tester à nouveau. Pour tracer/modifier les méthodes formulaire, il faut sortir du formulaire puis le réouvrir pour recharger la méthode correspondante. L'astuce est donc la suivante : lorsque vous effectuez le débogage complet d'un formulaire, placez le code que vous déboguez dans une méthode projet que vous utiliserez comme sous-routine à l'intérieur de la méthode formulaire. De cette manière, vous pouvez rester dans le formulaire, tracer, modifier et tester à nouveau votre formulaire car la sous-routine sera rechargée chaque fois qu'elle sera appelée par la méthode formulaire.

Bouton 'Enregistrer paramètres'

Ce bouton permet de sauvegarder la configuration courante de la fenêtre du débogueur (taille et position de la fenêtre, emplacement des lignes de division et contenu de la zone d'évaluation des expressions). Elle sera alors utilisée par défaut à chaque ouverture de la base. Ces paramétrages sont stockés dans le fichier de structure de la base.

Bouton 'Exécuter pas à pas'

La ligne courante de la méthode (indiquée par la flèche jaune — cette flèche s'appelle le **compteur de programme**) est exécutée et le débogueur passe à la ligne suivante. Le bouton **Exécuter pas à pas** ne passe pas dans les sous-routines et les fonctions. Il reste au niveau de la méthode que vous êtes en train de tracer. Si vous voulez également tracer les appels aux sous-routines et aux fonctions, utilisez le bouton **Pas à pas détaillé**.

Bouton 'Pas à pas détaillé'

Lors de l'exécution d'une ligne qui appelle une autre méthode (sous-routine ou fonction), ce bouton provoque l'affichage de la méthode appelée dans la fenêtre du débogueur, et permet au développeur de passer pas à pas dans cette méthode. La nouvelle méthode devient la méthode courante (en haut) dans la sous-fenêtre Chaîne d'appel de la fenêtre du débogueur. Lors de l'exécution d'une ligne qui n'appelle pas une autre méthode, ce bouton se comporte comme le bouton **Exécuter pas à pas**.

Bouton 'Pas à pas nouveau process'

Lors de l'exécution d'une ligne qui crée un nouveau process (par exemple qui appelle la commande Nouveau process) ce bouton ouvre une nouvelle fenêtre du débogueur qui vous permet de tracer la méthode de gestion du process que vous venez de créer. Lors de l'exécution d'une ligne qui ne crée pas de nouveau process, ce bouton se comporte comme le bouton **Exécuter pas à pas**.

Bouton 'Exécuter et sortir'

Si vous tracez des sous-routines et des fonctions, cliquer sur ce bouton vous permet d'exécuter l'intégralité de la méthode qui est en train d'être tracée, et de revenir à la méthode appelante. La fenêtre du débogueur retourne à la méthode précédente dans la chaîne d'appel. Si la méthode courante est la dernière méthode de la chaîne d'appel, la fenêtre du débogueur se referme.

Informations dans la barre d'outils de contrôle d'exécution

A la droite de la barre d'outils de contrôle d'exécution, le débogueur affiche les informations suivantes :
• Le nom de la méthode que vous êtes en train de tracer (en noir).
• La cause de l'ouverture du débogueur (en rouge).

Par exemple, dans la fenêtre affichée en tête de ce chapitre, vous pouvez voir les informations suivantes :
• La méthode Trace démo est la méthode qui est tracée.
• La fenêtre du débogueur s'affiche car il a rencontré un point d'arrêt.

Voici les causes qui provoquent l'apparition du débogueur et d'un message (en rouge) :
• **Commande TRACE** : un appel à TRACE a été émis.
• **Point d'arrêt atteint** : vous avez rencontré un point d'arrêt temporaire ou persistant.

• **Interruption demandée par l'utilisateur** : vous avez activé **Alt+Maj+clic droit** (sous Windows) ou **Control+Option+Commande+clic** (sous Mac OS) ou encore cliqué sur le bouton **Trace** dans la page **Process** de l'Explorateur d'exécution.
• **Détection d'un appel à : Nom de la commande** : Un appel à une commande 4D qui doit être interceptée est sur le point d'être exécuté.
• **Pas à pas nouveau process** : Vous avez cliqué sur le bouton **Pas à pas nouveau process** et ce message est affiché par la fenêtre du débogueur ouverte pour le process qui vient d'être créé.

Les sous-fenêtres du débogueur

La fenêtre du débogueur contient la barre de contrôle d'exécution décrite précédemment, ainsi que quatre sous-fenêtres redimensionnables :

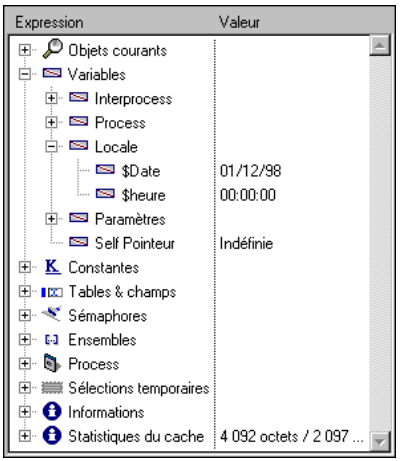
• La Fenêtre d'expression
• La Fenêtre de chaîne d'appel
• La Fenêtre d'évaluation
• La Fenêtre d'évaluation des méthodes

Les trois premières fenêtres affichent des listes hiérarchiques montrant l'information pertinente au débogage. La quatrième, la Fenêtre d'évaluation des méthodes, affiche le code source de la méthode tracée. Chaque fenêtre a un rôle précis pour vous assister dans le processus de débogage. Avec la souris, vous pouvez redimensionner la fenêtre du débogueur, ainsi que chaque sous-fenêtre. En outre, les trois premières sous-fenêtres sont séparées par une ligne pointillée dans le sens de la hauteur : vous pouvez redimensionner à votre convenance les colonnes à l'intérieur de chaque sous-fenêtre.

Référence

APPELER SUR ERREUR, Fenêtre d'erreur de syntaxe, Fenêtre d'évaluation, Fenêtre d'évaluation des méthodes, Fenêtre d'expression, Fenêtre de chaîne d'appel, Liste des points d'arrêt, Points d'arrêt sur commandes, Raccourcis du débogueur, TRACE, Un débogueur, pour quoi faire ?.

La **Fenêtre d'expression** est située en haut à gauche de la fenêtre du débogueur, sous la barre d'outils de contrôle d'exécution :



La Fenêtre d'expression affiche toutes les informations générales utiles sur l'environnement système, 4D et l'exécution du code.

La colonne **Expression** affiche le nom des objets et des expressions. La colonne **Valeur** affiche la valeur courante correspondant aux objets et expressions.

En cliquant sur une valeur dans la colonne de droite, vous pouvez modifier la valeur de l'objet, si cela est possible pour cet objet.

La liste hiérarchique multi-niveaux est organisée par thèmes au niveau supérieur. Les thèmes sont les suivants :

- Objets courants
- Variables
- Constantes
- Tables & champs
- Sémaphores
- Ensembles
- Process
- Sélections temporaires

• Informations
• Statistiques du cache

En fonction du thème, chaque article peut disposer d'un ou de plusieurs sous-niveaux. Pour déployer ou fermer chaque thème, il suffit de cliquer sur l'icône de déploiement située en face de son libellé. Si le thème comprend plusieurs niveaux d'information, il suffit de cliquer sur chaque icône pour explorer toutes les informations qu'il contient.

A tout moment, vous pouvez faire glisser un thème, un sous-thème ou un article, et le déposer dans la Fenêtre d'évaluation.

Objets courants

Ce thème affiche les valeurs des objets ou des expressions évaluables :
• utilisé(e)s dans la ligne de code à exécuter (celle qui est indiquée par le compteur de programme — la flèche jaune dans la fenêtre d'évaluation des méthodes),
• utilisé(e)s dans la ligne de code précédente.

Comme la ligne de code précédente est celle qui vient d'être exécutée, le thème Objets courants affiche donc de manière permanente les objets ou expressions de la ligne courante **avant et après** l'exécution de la ligne. Prenons l'exécution de la méthode suivante :

```
TRACE
a:=1
b:=a+1
c:=a+b
   ` ...
```

1. Vous entrez dans la fenêtre du débogueur avec le compteur de programme de la Fenêtre d'évaluation des méthodes placé à la ligne a:=1. A ce point précis, le thème Objets courants affiche :

```
a:      Indéfini
```

La variable a est affichée car elle est utilisée dans la ligne qui est sur le point d'être exécutée (mais elle n'a pas encore été initialisée).

2. Vous progressez d'une ligne. Le compteur de programme est maintenant positionné sur la ligne b:=a+1. A ce point, le thème Objets courants affiche :

```
a:      1
b:      Indéfini
```

La variable a s'affiche car elle est utilisée dans la ligne qui vient de s'exécuter, et qu'on lui avait assigné la valeur numérique 1. Elle s'affiche aussi parce qu'elle est utilisée dans la ligne qui sera exécutée, en tant qu'expression qui sera assignée à la variable b. La variable b s'affiche car elle est utilisée dans la ligne qui doit être exécutée (mais elle n'a pas encore été initialisée).

3. Vous progressez encore d'une ligne. Le compteur de programme est maintenant positionné sur la ligne c:=a+b. A ce point, le thème Objets courants affiche :

c: Indéfini
a: 1
b: 2

La variable c s'affiche car elle est utilisée dans la ligne à exécuter (mais elle n'a pas encore été initialisée). Les variables a et b sont affichées car elles étaient utilisées dans la ligne précédente et qu'elles le sont dans la ligne qui sera exécutée, etc.

Ce thème est extrêment pratique : chaque fois que vous exécutez une ligne, vous n'avez pas besoin de saisir une expression dans la fenêtre d'évaluation. Il vous suffit de surveiller les valeurs affichées par le thème Objets courants.

Variables
Ce thème se décompose en sous-thèmes :
• **Interprocess** : affiche la liste des variables interprocess en cours d'utilisation. Si vous n'utilisez pas de variable interprocess, la liste est vide. La valeur des variables interprocess peut être modifiée.
• **Process** : affiche la liste des variables process utilisées par le process courant. Cette liste est rarement vide. La valeur des variables process peut être modifiée.
• **Locales** : affiche la liste des variables locales utilisées par la méthode indiquée dans la sous-fenêtre d'évaluation de méthode. Cette liste peut être vide si aucune variable locale n'est utilisée ou si elles n'ont pas encore été créées. La valeur des variables locales peut être modifiée.
• **Paramètres** : affiche la liste des paramètres reçus par la méthode. Cette liste peut être vide si aucun paramètre n'a été passé à la méthode tracée. La valeur des paramètres peut être modifiée.
• **Self Pointeur** : affiche un pointeur vers l'objet courant si vous tracez une méthode objet. Cette valeur n'est pas modifiable.

Note : Vous pouvez modifier les variables et les champs de type Chaîne, Texte, numérique (Entier, Entier long, Réel), Date et Heure, c'est-à-dire les variables et les champs dont la valeur peut être saisie au clavier. Pour le type Entier vous pouvez utiliser indifféremment la notation décimale ou hexadécimale (par exemple, pour 256, vous pouvez taper '256' ou '0x100').

Les tableaux, comme les autres variables, apparaissent dans les sous-thèmes Interprocess, Process et Locales, en fonction de leur portée. Le débogueur affiche chaque tableau avec un niveau hiérarchique supplémentaire, ce qui vous permet d'obtenir ou de modifier les valeurs des éléments du tableau, s'il y en a. Le débogueur affiche les 100 premiers éléments (y compris l'élément zéro). La colonne Valeur affiche la taille du tableau en face de son nom. Une fois que vous avez déployé le tableau, le premier sous-article affiche le numéro de l'élément sélectionné courant, ensuite l'élément zéro, ensuite les autres éléments (jusqu'à 100) s'il y en a. Vous pouvez modifier les tableaux Alpha, Texte, Numérique et Date. Vous pouvez modifier le numéro de l'élément sélectionné, l'élément zéro et les autres éléments (jusqu'à 100) s'il y en a. Vous ne pouvez pas modifier la taille du tableau.

Note : A tout moment, vous pouvez glisser un article à partir de la Fenêtre d'expression vers la Fenêtre d'évaluation, y compris un élément de tableau.

Constantes : Ce thème affiche les constantes prédéfinies dans 4D, comme dans la page Constantes de la fenêtre de l'Explorateur. Les expressions de ce thème ne peuvent pas être modifiées.

Tables & champs : Ce thème affiche la liste des tables et des champs dans la base de données (à l'exception des sous-champs). Pour chaque table, la colonne Valeur affiche la taille de la sélection courante pour le process courant ainsi que (lorsque la ligne de la table est déployée) le nombre d'enregistrements verrouillés. Pour chaque champ, la colonne Valeur affiche la valeur du champ (à l'exception des images, sous-tables et BLOBs) pour l'enregistrement courant, s'il existe. Dans ce thème, les valeurs des champs peuvent être modifiées (notez qu'il n'y a alors pas d'annulation possible) mais pas celles de la table.

Sémaphores : Ce thème affiche la liste des sémaphores locaux dans les ensembles courants. Pour chaque sémaphore, la colonne Valeurs affiche le nom du process ayant posé le sémaphore. Si vous n'utilisez pas de sémaphore, la liste peut être vide. Les expressions de ce thème ne peuvent pas être modifiées. Il n'est pas possible de visualiser les sémaphores globaux.

Ensembles : Ce thème affiche la liste des ensembles définis dans le process courant (celui que vous êtes en train de tracer) ainsi que la liste des ensembles interprocess. La colonne Valeur affiche, pour chaque ensemble, le nombre d'enregistrements et le nom de la table. Si vous n'utilisez pas les ensembles, cette liste sera vide. Les expressions de ce thème ne peuvent pas être modifiées.

Process : Ce thème affiche la liste des process lancés depuis le début de la session de travail. La colonne Valeur affiche le temps déjà alloué à chaque process ainsi que son état (par exemple "En cours d'exécution", "Endormi", etc). Les expressions de ce thème ne peuvent pas être modifiées.

Sélections temporaires : Ce thème affiche la liste des sélections temporaires process définies dans le process courant (celui que vous êtes en train de tracer) ainsi que la liste des sélections temporaires interprocess. Pour chaque sélection temporaire, la colonne Valeur affiche le nombre d'enregistrements et le nom de la table. Si vous n'utilisez pas les sélections temporaires, cette liste sera vide. Les expressions de ce thème ne peuvent pas être modifiées.

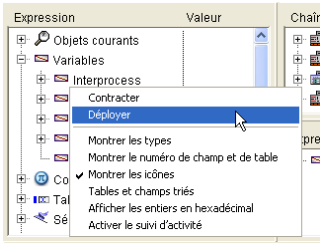
Informations : Ce thème affiche les informations générales, comme la table par défaut courante (s'il y en a une). Les expressions contenues dans ce thème ne peuvent pas être modifiées.

Statistiques du cache : Ce thème affiche diverses statistiques concernant l'utilisation des tables d'adresses et d'index, des pages d'index et des sélections temporaires chargées dans le cache de 4D. Les expressions contenues dans ce thème ne peuvent pas être modifiées.

Menu contextuel de la fenêtre d'expression

Le menu contextuel de la fenêtre d'expression vous propose des options supplémentaires. Pour afficher ce menu il vous suffit de :
• Sous Windows, cliquer n'importe où dans la fenêtre d'expression avec le **bouton droit** de la souris
• Sous Mac OS, utiliser la combinaison **Control+clic** n'importe où dans la fenêtre d'expression.

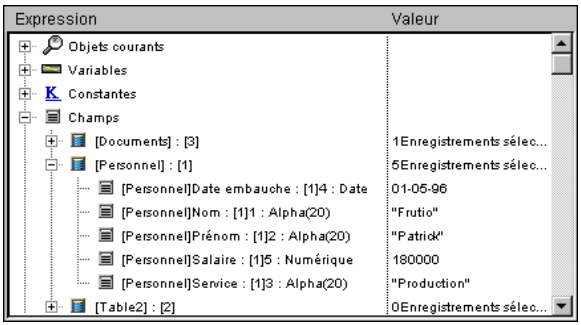
Voici le menu contextuel de la fenêtre d'expression :



• **Contracter** : Contracte tous les niveaux de la liste hiérarchique des expressions.
• **Déployer** : Déploie tous les niveaux de la liste hiérarchique des expressions.
• **Montrer les types** : Lorsque vous sélectionnez cette option, le type de l'objet s'affiche en face de l'objet (lorsque cela est pertinent).
• **Montrer le numéro de champ et de table** : Si vous travaillez avec le numéro des tables ou de champs, ou avec des pointeurs utilisant les commandes Table ou Champ, cette option est très pratique : en face de chaque table et champ, elle affiche le numéro de la table ou du champ.

• **Montrer les icônes** : Chaque objet est précédé d'une icône qui indique son type. Vous pouvez désactiver cette option pour accélérer l'affichage, ou tout simplement parce que l'option **Montrer les types** vous convient.
• **Tables et champs triés** : Cette option force les tables et les champs à s'afficher par ordre alphabétique (dans leurs listes respectives).
• **Afficher les entiers en hexadécimal** : Les nombres s'affichent en notation décimale. Sélectionnez cette option pour les afficher en hexadécimal.
• **Activer le suivi d'activité** : Active le suivi d'activité (contrôle avancé de l'activité interne de l'application) et affiche les informations collectées dans des thèmes supplémentaires, **Séquenceur**, **Web** et **Réseau**.

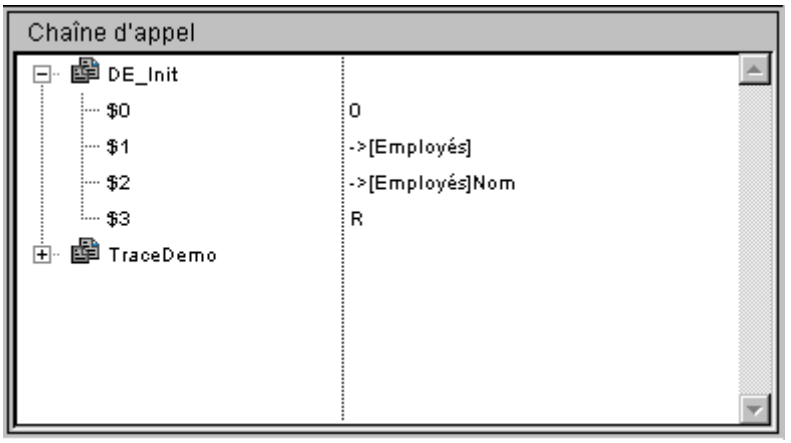
Ci-dessous, la fenêtre d'expression telle qu'elle se présente lorsque vous sélectionnez toutes les options :



Référence

Débogueur, Fenêtre d'évaluation, Fenêtre d'évaluation des méthodes, Fenêtre de chaîne d'appel, Raccourcis du débogueur.

Une méthode peut appeler d'autres méthodes, qui à leur tour peuvent appeler d'autres méthodes. Pour cette raison, il est très utile d'avoir sous les yeux, pendant le débogage, la chaîne des méthodes, ou **chaîne d'appel**. Cette chaîne peut être visualisée dans la fenêtre située en haut et à droite du débogueur. Les méthodes y sont affichées de manière hiérarchique :



• Chaque niveau principal est le nom d'une méthode. L'élément placé en tête de la liste est la méthode que vous êtes en train de tracer, le niveau suivant est le nom de la méthode appelante (la méthode qui a appelé celle que vous êtes en train de tracer), le niveau suivant est l'appelant de la méthode appelante, etc. Dans l'exemple ci-dessus, la méthode DE_Init est tracée. Elle a été appelée par la méthode TraceDemo.

• Lorsque vous double-cliquez sur le nom d'une méthode dans la fenêtre de chaîne d'appel, vous basculez sur la méthode appelante dont le code source est affiché dans la fenêtre d'évaluation de méthodes. Vous pouvez ainsi voir rapidement comment la méthode appelante a effectué son appel à la méthode appelée. Vous pouvez aussi examiner toutes les étapes de la chaîne d'appel.

• Lorsque vous cliquez sur l'icône de déploiement jouxtant le nom d'une méthode, vous déployez ou vous contractez la liste des paramètres ($1, $2...) ainsi que le résultat ($0) optionnel d'une fonction. La valeur s'affiche à droite de la fenêtre. En cliquant sur une valeur quelconque à droite, vous pouvez changer la valeur du résultat ou de tout paramètre. Dans l'illustration ci-dessus :

1. DE_Init $0 est actuellement indéfinie car la méthode n'a assigné aucune valeur à $0 (parce qu'elle n'a pas encore exécuté cette affectation, ou parce que la méthode est une sous-routine et non une fonction).
2. DE_Init a reçu trois paramètres de TraceDemo. $1 est un pointeur vers la table [Employés], $2 est un pointeur vers le champ [Employés]Nom et $3 est un paramètre alphanumérique de valeur "R".

• Lorsque vous avez déployé la liste des paramètres/résultats d'une méthode, vous pouvez également les faire glisser vers la Fenêtre d'évaluation.

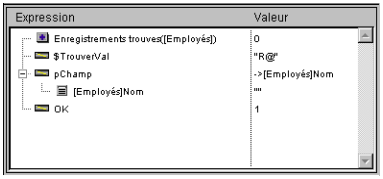
Référence

Débogueur, Fenêtre d'évaluation, Fenêtre d'évaluation des méthodes, Fenêtre d'expression, Raccourcis du débogueur.

Juste au-dessous de la Fenêtre de chaîne d'appel se trouve la Fenêtre d'évaluation. Cette fenêtre sert à évaluer les expressions. Vous pouvez évaluer tout type d'expression, y compris les champs, les variables, les pointeurs, les calculs, les fonctions intégrées, vos propres fonctions, et tout ce qui retourne une valeur.

Vous pouvez évaluer toute expression qui peut être affichée sous forme de texte. Cela ne s'applique donc pas aux champs ni aux variables image ou BLOB. En revanche, lorsque vous déployez les listes hiérarchiques, le débogueur vous permet d'afficher les tableaux et les pointeurs. Pour afficher le contenu des BLOBs, vous pouvez utiliser les commandes sur les BLOBs, comme par exemple BLOB vers texte.

Dans l'exemple ci-dessous, vous pouvez voir plusieurs expressions : deux variables, un pointeur vers un champ, le résultat d'une fonction intégrée et un calcul.



Insérer une nouvelle expression

Vous pouvez ajouter une expression à évaluer dans la fenêtre d'expression de l'une des manières suivantes :
- Glissez-déposez un objet ou une expression à partir de la Fenêtre d'expression ;
- Glissez-déposez un objet ou une expression à partir de la Fenêtre de chaîne d'appel ;
- Cliquez sur une expression évaluable dans la Fenêtre d'évaluation des méthodes.

En outre, pour créer une expression vide, double-cliquez n'importe où dans l'espace vide de la fenêtre d'évaluation. Cela crée une expression vide ` Nouvelle expression prête à l'édition. Vous pouvez saisir toute formule 4D qui retourne un résultat.

Dès que vous avez saisi la formule, appuyez sur la touche **Entrée** ou **Retour chariot** (ou cliquez n'importe où dans la fenêtre) pour obtenir l'évaluation de l'expression.

Si vous voulez changer d'expression, cliquez dessus pour la sélectionner, et cliquez de nouveau pour entrer en mode édition.

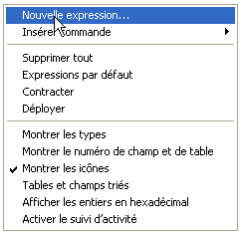
Si vous n'avez plus besoin d'une expression, cliquez dessus pour la sélectionner, puis appuyez sur la touche **Retour Arrière** ou **Suppr**.

Attention : Soyez prudent lorsque vous évaluez une expression 4D modifiant la valeur d'une des Variables système (par exemple la variable OK). En effet, dans ce cas l'exécution du reste de la méthode peut être faussée.

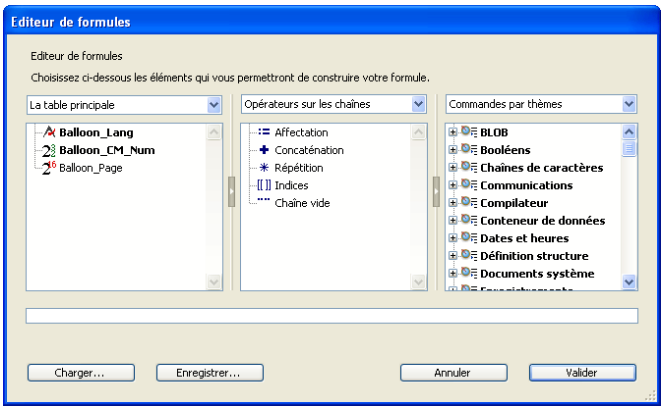
Menu contextuel de la fenêtre d'évaluation

Le menu contextuel de la Fenêtre d'évaluation vous permet d'accéder à l'éditeur de formules de 4D, pour vous aider à saisir et éditer une expression. Le menu contextuel vous propose également des options supplémentaires.

Pour obtenir ce menu, cliquez n'importe où dans la fenêtre d'évaluation avec le **bouton droit** de la souris.



• **Nouvelle expression** : Cette commande insère une nouvelle expression et affiche l'éditeur de formules de 4D (voir ci-dessous) dans lequel vous pouvez saisir la nouvelle expression.



Pour plus d'informations sur l'éditeur de formules, reportez-vous au manuel *Mode Développement* de 4D.

• **Insérer commande** : Cette commande est un raccourci pour placer une commande (sans utiliser l'éditeur de formules) en tant que nouvelle expression.
• **Supprimer tout** : Efface toutes les expressions présentes.
• **Expressions par défaut** : Recopie la liste d'objets de la zone Expression.
• **Contracter/Déployer** : Contracte ou déploie toutes les expressions dont l'évaluation se fait en utilisant les listes hiérarchiques (pointeurs, tableaux, etc.).
• **Montrer les types** : Si cette option est sélectionnée, le type de l'objet s'affiche en face de chaque objet (lorsque c'est pertinent).
• **Montrer le numéro de champ et de table** : Cette commande est extrêmement pratique si vous travaillez avec des numéros de table ou de champs, ou avec des pointeurs utilisant des commandes comme Table ou Champ. En face de chaque table et champ, elle affiche le numéro de la table ou du champ.
• **Montrer les icônes** : Chaque objet est précédé par une icône qui symbolise son type. Vous pouvez désactiver cette option pour accélérer l'affichage, ou simplement parce que l'option **Montrer les types** vous convient.
• **Tables et champs triés** : Cette option force l'affichage des tables et des champs par ordre alphabétique (dans leurs listes respectives).
• **Afficher les entiers en hexadécimal** : Par défaut, les nombres entiers sont affichés en notation décimale. Sélectionnez cette option si vous souhaitez un affichage hexadécimal.
• **Activer le suivi d'activité** : Active et affiche les informations de suivi d'activité (cf. Fenêtre d'expression).

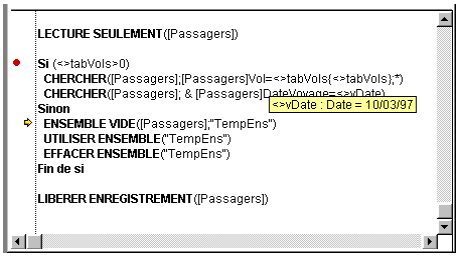
Référence

Débogueur, Fenêtre d'évaluation des méthodes, Fenêtre d'expression, Fenêtre de chaîne d'appel, Raccourcis du débogueur.

La Fenêtre d'évaluation des méthodes affiche le code source de la méthode en train d'être tracée.

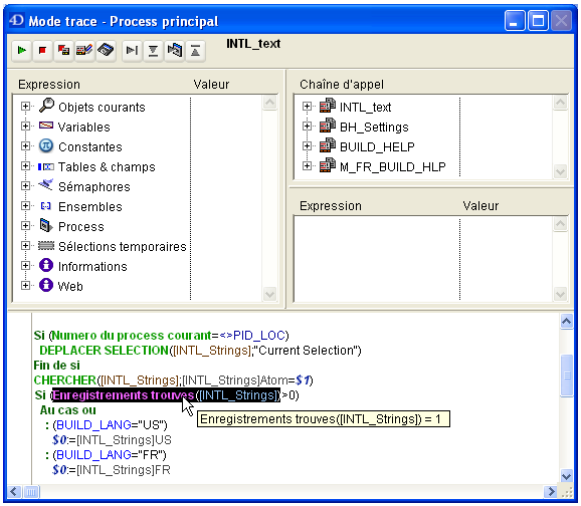
• Si la méthode est trop longue pour tenir dans la zone de texte, vous pouvez la faire défiler.

• Lorsque vous positionnez le pointeur de la souris au-dessus d'une expression qui peut être évaluée (champ, variable, pointeur, tableau,...) le débogueur affiche une **Info-bulle** qui indique la valeur courante de l'objet ou de l'expression, et son type déclaré. Par exemple :



Une info-bulle apparaît lorsque le pointeur de la souris est positionné sur la variable <>vDate. Dans cet exemple, c'est une variable de type date contenant la valeur suivante : 10/03/97.

• Vous pouvez également sélectionner une portion de texte dans la zone affichant le code en cours d'exécution. Dans ce cas, lorsque le curseur de la souris est placé au-dessus du texte sélectionné, l'info-bulle fournit la valeur de la sélection :



Lorsque vous cliquez sur un nom de variable ou de champ, il est automatiquement sélectionné.

Astuce : Vous pouvez copier instantanément dans la Fenêtre d'évaluation l'expression ou l'objet sélectionné. Vous disposez de trois solutions :
• par simple glisser-déposer : cliquez sur le texte sélectionné, faites-le glisser et relâchez-le dans la zone d'évaluation.
• cliquer sur le texte sélectionné tout en maintenant enfoncée la touche **Ctrl** (Windows) ou **Commande** (Mac OS).
• utiliser la combinaison de touches **Ctrl+D** (Windows) ou **Commande+D** (Mac OS).

Compteur de programme

Une flèche jaune, dans la marge gauche de la fenêtre d'évaluation des méthodes (voir illustration ci-dessus) indique la prochaine ligne à être exécutée. Cette flèche s'appelle le **compteur de programme**. Elle indique toujours la ligne sur le point d'être exécutée.

Pendant le débogage, vous pouvez **déplacer** le compteur de programme. Il vous suffit de cliquer sur la flèche jaune et de la faire glisser devant la ligne que vous désirez.

ATTENTION : Utilisez cette fonction avec précaution ! Déplacer vers l'avant le compteur de programme ne signifie pas que le débogueur exécute en même temps les lignes sur lesquelles vous passez. De la même manière, le faire remonter ne signifie pas que le débogueur inverse l'effet des lignes qui ont déjà été exécutées.

Lorsque vous déplacez le compteur de programme, vous indiquez simplement au débogueur de "continuer à tracer et exécuter à partir de cet endroit ". Tous les paramètres, champs, variables, etc. courants, ne sont pas affectés par le déplacement.

Voici un exemple de déplacement du compteur de programme. Imaginons que vous êtes en train de déboguer le code suivant :

```
    ` ...
Si (Cette condition)
    FAIRE QUELQUE CHOSE
Sinon
    FAIRE AUTRE CHOSE
Fin de si
```

Le compteur de programme est placé à la ligne Si(Cette condition). Vous avancez d'un pas, et voyez que le compteur se place à la ligne FAIRE AUTRE CHOSE. Pas de chance, car vous vouliez en fait exécuter la première partie de l'alternative. Dans ce cas, et dans la mesure où l'expression Cette condition n'effectue pas d'opérations affectant les étapes suivantes de votre test, remontez le compteur à la ligne FAIRE QUELQUE CHOSE, et vous êtes prêt à continuer à tracer la portion de code qui vous intéresse.

Définir des points d'arrêt dans le débogueur

Pendant le débogage, vous pouvez avoir besoin de sauter certaines portions de votre code. Le débogueur vous permet d'utiliser plusieurs méthodes pour exécuter votre code **jusqu'à un certain point** :

• Pendant que vous exécutez au pas à pas, vous pouvez cliquer sur le bouton **Exécuter pas à pas** au lieu de **Pas à pas détaillé**, lorsque vous ne voulez pas entrer dans les éventuelles sous-routines ou fonctions appelées dans la ligne du compteur de programme.
• Si vous entrez par erreur dans une sous-routine, vous pouvez l'exécuter et revenir directement à la méthode appelante en cliquant sur le bouton **Exécuter et sortir**.
• Si vous appelez la commande TRACE quelque part, vous pouvez cliquer sur le bouton **Pas de Trace** pour reprendre l'exécution jusqu'à cet appel TRACE.

Par exemple, imaginons que vous soyez en train d'exécuter le code suivant. Le compteur de programme est placé devant la ligne TOUT SELECTIONNER ([CetteTable]) :

```
    ` ...
TOUT SELECTIONNER([CetteTable])
$vrResult:=0
Boucle($vlEnregist;1;Enregistrements trouves([CetteTable]))
    $vrResult:=CetteaFonction([CetteTable])
    ENREGISTREMENT SUIVANT([CetteTable])
Fin de boucle
Si ($vrResult>=$vrLimiteVal)
    ` ...
```

Vous voulez évaluer la valeur de $vrResult à la fin de la boucle Boucle. Comme cela peut prendre un certain temps d'exécution pour atteindre cette portion de votre code, vous voulez abandonner l'exécution courante puis éditer la méthode pour insérer un appel TRACE avant la ligne Si ($vrResult...

Vous pourriez aussi exécuter la boucle, mais si la table [CetteTable] contient plusieurs centaines d'enregistrements, cette opération vous prendra la journée. Pour éviter des situations de ce type, le débogueur met à votre disposition des **points d'arrêt**. Vous insérez des points d'arrêt en cliquant dans la marge gauche de la fenêtre d'évaluation des méthodes. Par exemple, vous cliquez dans la marge gauche de la fenêtre, au niveau de la ligne Si ($vrResult... :

Vous insérez un point d’arrêt à la ligne marquée par un point rouge. Ensuite, vous cliquez sur le bouton **Pas de Trace**.
Vous reprenez l’exécution normale **jusqu’à** la ligne marquée par le point d’arrêt. Cette ligne n’est pas exécutée : vous êtes revenu au mode Trace. Dans cet exemple, toute la boucle a été exécutée normalement, et, une fois arrivé au point d’arrêt, il vous suffit de placer le curseur de la souris au-dessus de vrResult pour évaluer sa valeur à la sortie de la boucle.
Si vous placez un point d’arrêt au-delà du compteur de programme et que vous cliquez sur le bouton **Pas de Trace**, vous éviterez de tracer des parties de la méthode.

Note : Vous pouvez également définir des points d'arrêt directement dans l'éditeur de méthodes de 4D. Reportez-vous à la section Points d'arrêt.

Un **point d’arrêt rouge** est un point d’arrêt **persistant**. Une fois que vous l’avez créé, il reste, même lorsque vous quittez la base et la réouvrez par la suite.

Pour le supprimer, vous pouvez procéder d'une des manières suivantes :
• Si vous avez fini de l’utiliser, cliquez dessus, et le point d’arrêt disparaît.
• Si vous n’avez pas fini de l’utiliser, mais que vous voulez le désactiver provisoirement, il vous faut l’éditer. Reportez-vous pour ceci à la section Points d’arrêt.

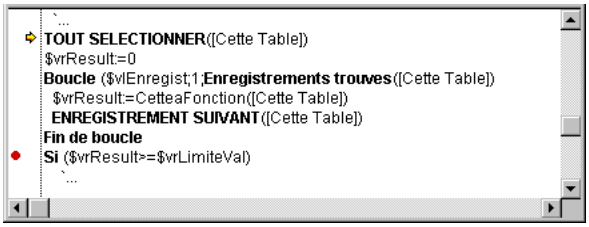
Référence

Débogueur, Fenêtre d'évaluation, Fenêtre d'expression, Fenêtre de chaîne d'appel, Points d'arrêt.

Comme décrit dans la section Fenêtre d'évaluation des méthodes, vous définissez un point d'arrêt en cliquant dans la marge gauche de la fenêtre d'évaluation des méthodes ou de la fenêtre d'édition des méthodes, au niveau de la ligne de code sur laquelle vous voulez vous arrêter.

Note : L'insertion et l'édition de points d'arrêt peuvent être effectuées indifféremment dans l'éditeur de méthodes ou le débogueur. Il y a une interaction dynamique entre les deux éditeurs : un point d'arrêt inséré ou modifié dans un éditeur est immédiatement reporté dans l'autre (ainsi que dans la Liste des points d'arrêt de l'explorateur d'exécution). A noter toutefois que les points d'arrêt **provisoires** ne peuvent être créés que dans le débogueur (cf. ci-dessous).

Dans l'exemple suivant, un point d'arrêt a été placé, dans le débogueur, en regard de la ligne Si ($vrResult>=$vrLimitVal) :



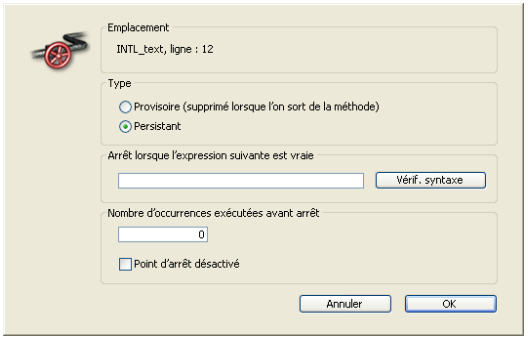
Si vous cliquez de nouveau sur la puce rouge, vous supprimez le point d'arrêt.

Editer un point d'arrêt

La combinaison **Alt+clic** (sous Windows) ou **Option+clic** (sous Mac OS) dans la marge gauche de la fenêtre d'évaluation des méthodes (ou de l'éditeur de méthodes) ouvre la **fenêtre des propriétés du point d'arrêt**.

- Si vous avez cliqué sur un point d'arrêt existant, la fenêtre s'ouvre pour ce point d'arrêt.
- Si vous cliquez sur une ligne où aucun point d'arrêt n'a été placé, un point d'arrêt est créé et sa fenêtre de propriétés s'affiche.

Voici la **fenêtre des propriétés du point d'arrêt** :



• **Emplacement** : Vous indique le nom de la méthode et le numéro de la ligne où le point d'arrêt a été placé. Vous ne pouvez pas modifier cette information.

• **Type** : Par défaut, vous créez des points d'arrêt **persistants**, symbolisés par une puce rouge. Toutefois, le débogueur vous permet de définir des **points d'arrêt provisoires**. Il est utile de pouvoir disposer de points d'arrêt provisoires lorsque vous voulez vous arrêter une seule fois dans une méthode. Un point d'arrêt provisoire est identifié dans la fenêtre d'évaluation des méthodes du débogueur par une **puce verte**. Vous pouvez poser un point d'arrêt provisoire directement dans la fenêtre d'évaluation des méthodes en cliquant dans la marge gauche tout en maintenant les touches **Alt** + **Majuscule** (Windows) ou **Option** + **Majuscule** (Macintosh) enfoncées.

Note : Les points d'arrêt provisoires peuvent être définis dans le débogueur uniquement.

• **Arrêt lorsque l'expression suivante est vraie** : Saisissez une formule 4D qui retourne Vrai ou Faux. Si, par exemple, vous voulez ne vous arrêter à une ligne que lorsque, par exemple, Enregistrements dans selection([aTable])=0, saisissez cette formule et l'arrêt se produira uniquement si vous n'avez aucun enregistrement sélectionné pour la table [aTable] lorsque le débogueur rencontrera la ligne où le point d'arrêt est placé. Si vous n'êtes pas sûr de la syntaxe de votre formule, cliquez sur le bouton **Vérif. Syntaxe**.

• **Nombre d'occurrences exécutées avant arrêt** : Vous pouvez insérer un point d'arrêt sur une ligne de code placée dans une structure en boucle (Tant que, Repeter, Boucle) ou placée dans une sous-routine ou fonction appelée à partir d'une boucle. Si vous savez que le problème que vous cherchez n'arrivera pas avant au moins la 200e itération de la boucle, saisissez 200 et le point d'arrêt s'activera à la 201e itération.

• **Point d'arrêt désactivé** : Vous avez créé un point d'arrêt persistant dont vous n'avez plus besoin pour le moment mais qui pourrait vous servir plus tard. Il vous suffit dans ce cas de le désactiver. Un point d'arrêt désactivé s'affiche sous forme de tiret (-) et non plus de puce (•) à la fois dans le débogueur, dans l'éditeur de méthodes et dans la Liste des points d'arrêt.

Les points d'arrêt peuvent être créés et édités dans la fenêtre du débogueur et l'éditeur de méthodes. Mais vous pouvez également éditer des points d'arrêt existants en affichant la liste des points d'arrêt, dans la page "Point d'arrêt" de l'Explorateur d'exécution. Pour plus d'informations, reportez-vous à la section Liste des points d'arrêt.

Référence

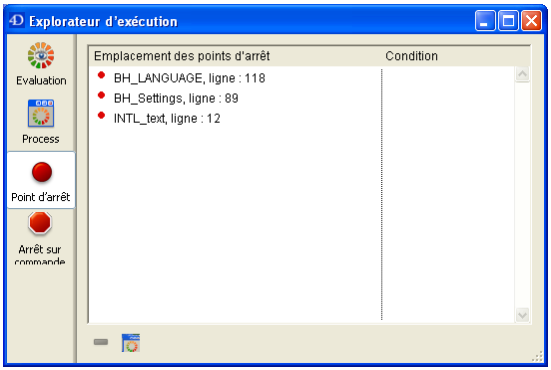
Débogueur, Fenêtre d'évaluation des méthodes, Liste des points d'arrêt, Points d'arrêt sur commandes.

La liste des points d'arrêt est une page de l'Explorateur d'exécution qui vous permet de gérer les points d'arrêt que vous avez créés dans la fenêtre du débogueur ou dans l'éditeur de méthodes.

Pour afficher la Liste des points d'arrêt, procédez de la manière suivante :

1. Choisissez **Explorateur d'exécution** dans le menu **Exécution**.
La fenêtre de l'Explorateur d'exécution peut être affichée sous forme de palette flottante (pour cela, maintenez la touche **Maj** enfoncée au moment de sélectionner la commande de menu). L'Explorateur d'exécution est alors disponible dans tous les environnements de 4D. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

La fenêtre de l'Explorateur d'exécution apparaît.
2. Cliquez sur le bouton **Points d'arrêt** pour afficher la liste des points d'arrêt :



La liste des points d'arrêts est divisée en deux colonnes :
• La colonne de gauche indique le statut actif/inactif du point d'arrêt, suivi du nom de la méthode et du numéro de la ligne à laquelle le point d'arrêt a été placé dans la fenêtre du débogueur ou de l'éditeur de méthodes.
• La colonne de droite affiche la condition optionnellement associée au point d'arrêt.

Vous pouvez définir dans cette fenêtre une condition pour un point d'arrêt.
Vous pouvez activer, désactiver ou supprimer chaque point d'arrêt. En revanche, vous ne pouvez pas ajouter de point d'arrêt dans l'Explorateur. Les points d'arrêt persistants ne peuvent être créés qu'à partir de la fenêtre du débogueur ou de l'éditeur de méthodes.
Vous pouvez également ouvrir une fenêtre de l'éditeur de méthodes affichant la méthode à laquelle est associé le point d'arrêt en double-cliquant sur le libellé du point d'arrêt.

Définir une condition pour un point d'arrêt

Pour définir une condition pour un point d'arrêt, procédez de la manière suivante :
1. Cliquez dans la colonne de droite en regard du point d'arrêt.
2. Saisissez une formule 4D (expression ou appel de commande ou méthode projet) retournant une valeur booléenne (Vrai ou Faux).

Note : Pour supprimer une condition, effacez la formule correspondante.

Activer/Désactiver un point d'arrêt

Pour activer ou désactiver un point d'arrêt, procédez de la manière suivante :
1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste.
2. Sélectionnez la commande **Activer/Désactiver** dans le menu contextuel.

Raccourci : Chaque point d'arrêt dans la liste peut être activé/désactivé par un simple clic sur la puce (•) placée devant son libellé. La puce se transforme en tiret (-) lorsque le point d'arrêt est désactivé.

Supprimer un point d'arrêt

Pour supprimer un point d'arrêt, procédez de la manière suivante :
1. Sélectionnez le libellé du point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste.
2. Appuyez sur la touche **Retour arrière**, ou cliquez sur le bouton **Supprimer** au-dessous de la liste.

Note : Pour supprimer tous les points d'arrêt, cliquez sur le bouton **Supprimer tout** (deuxième bouton situé sous la liste), ou encore sélectionnez la commande **Supprimer tout** dans le menu contextuel.

Astuces

• L'ajout de conditions aux points d'arrêt persistants ralentit l'exécution du code car la condition est évaluée à chaque fois que le point d'arrêt est rencontré. D'un autre côté, le débogage est plus rapide car 4D ignore automatiquement les occurrences qui ne satisfont pas aux conditions.

• La désactivation d'un point d'arrêt persistant produit quasiment le même effet que sa suppression. Lors de l'exécution du code, le débogueur ne passe presque pas de temps sur le point d'arrêt. L'avantage de la désactivation est que vous n'avez pas à redéfinir le point d'arrêt en cas de besoin.

Référence

Débogueur, Fenêtre d'évaluation des méthodes, Points d'arrêt, Points d'arrêt sur commandes, Un débogueur, pour quoi faire ?.

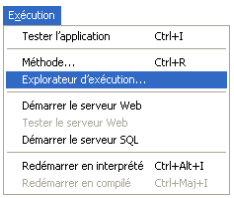
La liste des Points d'arrêt sur commandes est une page de l'Explorateur d'exécution qui vous permet d'ajouter des points d'arrêt supplémentaires dans votre code en interceptant des appels aux commandes 4D.

Placer un point d'arrêt sur une commande vous permet de commencer à tracer l'exécution de n'importe quel process dès qu'une commande particulière est appelée par le process. A la différence d'un point d'arrêt placé dans une méthode projet (qui, par conséquent, déclenche le mode trace uniquement lorsqu'il est atteint), l'aire d'action d'un point d'arrêt sur commande comprend tous les process qui exécutent du code 4D et qui appellent cette commande.

Placer un point d'arrêt sur une commande est un moyen pratique de tracer des grandes parties du code sans devoir insérer des points d'arrêt à des emplacements arbitraires. Si, par exemple, l'exécution dans votre code de plusieurs process durant un certain laps de temps provoque l'effacement d'un enregistrement qui ne devrait théoriquement pas être supprimé, vous pouvez limiter le champ d'investigation en plaçant un point d'arrêt sur les commandes telles que SUPPRIMER ENREGISTREMENT et SUPPRIMER SELECTION. A chaque fois que ces commandes sont appelées, vous pouvez vérifier si l'enregistrement en question a été supprimé ou non, et donc isoler la partie fautive du code.
Bien entendu, l'expérience aidant, vous pourrez combiner l'utilisation de points d'arrêt dans les méthodes et sur des commandes.

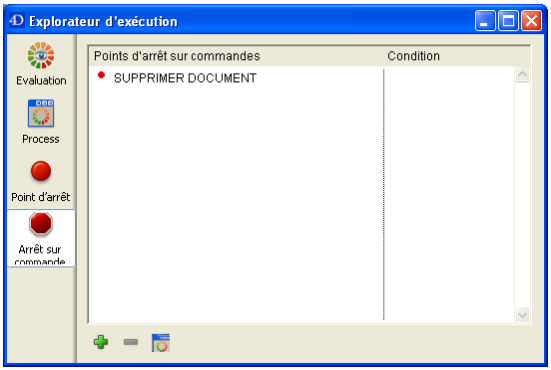
Pour afficher la liste des Points d'arrêt sur commandes, procédez de la manière suivante :

1. Choisissez **Explorateur d'exécution** dans le menu **Exécution**.
La fenêtre de l'Explorateur d'exécution peut être affichée sous forme de palette flottante (pour cela, maintenez la touche **Maj** enfoncée au moment de sélectionner la commande de menu). L'Explorateur d'exécution est alors disponible dans tous les environnements de 4D. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.



La fenêtre de l'Explorateur d'exécution apparaît.

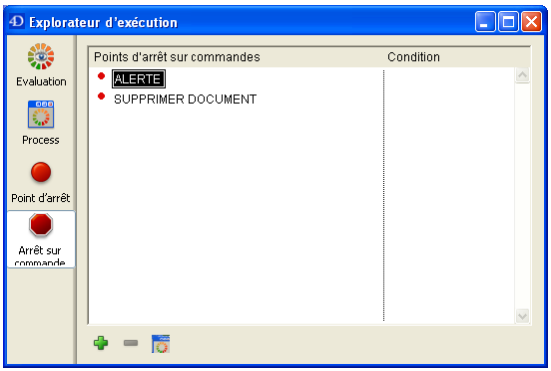
2. Cliquez sur le bouton **Arrêt sur commande** pour afficher la liste des points d'arrêt sur commande :



Cette page liste les commandes à intercepter au moment de leur exécution. Elle est divisée en deux colonnes :
• La colonne de gauche indique le statut actif/inactif du point d'arrêt, suivi du nom de la commande.
• La colonne de droite affiche la condition optionnellement associée au point d'arrêt.

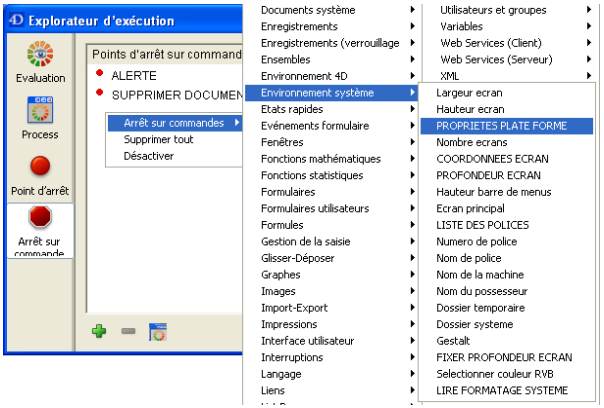
Ajouter un nouveau point d'arrêt sur commande

Pour ajouter un nouveau point d'arrêt sur une commande, vous disposez de plusieurs possibilités :
• cliquer sur le bouton d'ajout (en forme de +) situé au-dessous de la liste.
• double-cliquer directement dans la liste.
Dans ces deux cas, une nouvelle entrée est ajoutée dans la liste, avec la commande ALERTE par défaut.



Le libellé ALERTE est automatiquement placé en mode édition, vous pouvez alors saisir le nom de la commande à laquelle vous souhaitez associer un point d'arrêt. Une fois que vous avez terminé, appuyez sur la touche **Entrée** ou **Retour chariot** pour valider votre choix.

• cliquer dans la liste avec le **bouton droit** de la souris pour afficher un menu contextuel. Sélectionnez **Arrêt sur commande**, le menu liste alors toutes les commandes 4D :



Choisissez votre commande parmi les sous-menus des thèmes puis des noms de commandes. Une nouvelle entrée est ajoutée, affichant le nom de la commande sélectionnée.

Modifier un point d'arrêt sur commande

Pour modifier un point d'arrêt sur une commande, procédez de la manière suivante :
1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste (s'il n'y a pas déjà un point d'arrêt sélectionné et en mode édition).
2. Pour passer un point d'arrêt du mode sélection au mode édition et inversement, appuyez sur la touche **Entrée** ou **Retour chariot**.
3. Saisissez ou modifiez le nom de la commande.
4. Pour valider vos modifications, appuyez sur la touche **Entrée** ou **Retour chariot**. Si le nom que vous avez saisi ne correspond pas à une commande 4D existante, le point d'arrêt affiche la valeur précédente. Dans le cas d'une nouvelle entrée, le libellé ALERTE est réaffiché.

Activer/Désactiver un point d'arrêt sur commande

Pour activer ou désactiver un point d'arrêt sur commande, procédez de la manière suivante :
1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste (s'il n'y pas déjà un point d'arrêt sélectionné et en mode édition).
2. Choisissez la commande **Activer/Désactiver** dans le menu contextuel.

Raccourci : Chaque point d'arrêt dans la liste peut être activé/désactivé par un simple clic sur la puce (•) placée devant son libellé. La puce se transforme en tiret (–) lorsque le point d'arrêt est désactivé.

Supprimer un point d'arrêt sur commande

Pour supprimer un point d'arrêt sur commande, procédez de la manière suivante :
1. Sélectionnez le point d'arrêt en cliquant dessus ou en utilisant les touches fléchées pour naviguer dans la liste (s'il n'y pas déjà un point d'arrêt sélectionné et en mode édition).

2. Assurez-vous que le point d'arrêt se trouve en mode sélection (appuyez sur la **Entrée** ou **Retour chariot** pour passer alternativement du mode édition au mode sélection)
3. Appuyez sur la touche **Retour arrière** ou cliquez sur le bouton de suppression (en forme de '-') situé sous la liste.

Note : Pour supprimer tous les points d'arrêt, cliquez sur le bouton **Supprimer tout** (troisième bouton placé sous la liste) ou choisissez la commande **Supprimer tout** dans le menu contextuel.

Définir une condition pour un point d'arrêt sur commande

Pour définir une condition pour un point d'arrêt sur commande, procédez de la manière suivante :
1. Cliquez dans la colonne de droite en regard du point d'arrêt.
2. Saisissez une formule 4D (expression ou appel de commande ou méthode projet) retournant une valeur booléenne (Vrai ou Faux).

Note : Pour supprimer une condition, effacez la formule correspondante.

Astuces

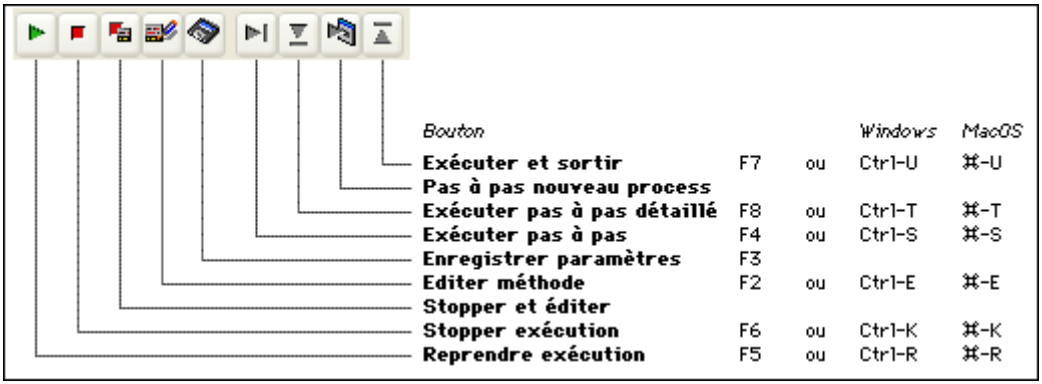
• L'ajout de conditions aux points d'arrêt sur commande ralentit l'exécution du code car la condition est évaluée à chaque fois que le point d'arrêt est rencontré. D'un autre côté, le débogage est plus rapide car 4D ignore automatiquement les occurences qui ne satisfont pas aux conditions.

• La désactivation d'un point d'arrêt sur commande produit quasiment le même effet que sa suppression. Lors de l'exécution du code, le débogueur ne passe presque pas de temps sur le point d'arrêt. L'avantage de la désactivation est que vous n'avez pas à redéfinir le point d'arrêt en cas de besoin.

Ce chapitre détaille les raccourcis disponibles dans la fenêtre du débogueur.

Barre d'outils de contrôle d'exécution

→ L'illustration ci-dessous présente les raccourcis des boutons situés dans la partie supérieure gauche de la fenêtre du débogueur :



→ La combinaison Maj+F5 ou Maj+clic sur le bouton **Reprendre exécution** reprend l'exécution et en plus désactive tous les appels TRACE ultérieurs dans le process courant.

Sous-fenêtre zone d'expression

→ un clic avec le bouton droit de la souris (Windows) ou Control+clic (Macintosh) dans la sous-fenêtre d'expression déroule le menu contextuel.
→ un double-clic sur un article de la fenêtre d'expression copie cet article dans la fenêtre d'évaluation.

Sous-fenêtre de chaîne d'appel

→ Un double-clic sur le nom d'une méthode dans la Fenêtre de chaîne d'appel affiche la méthode dans la sous-fenêtre d'évaluation des méthodes, à la ligne correspondant à la chaîne d'appel.

Sous-fenêtre d'évaluation

→ Un clic avec le bouton droit de la souris (Windows) ou Control+Clic (Macintosh) dans la fenêtre d'évaluation déroule le menu contextuel de la fenêtre.
→ Un double-clic dans la sous-fenêtre d'évaluation crée une nouvelle expression.

Fenêtre d'évaluation des méthodes

→ Un clic dans la marge gauche place ou supprime un point d'arrêt.
→ Alt+Majuscule+clic (Windows) ou Option+Majuscule+clic (Macintosh) pose un point d'arrêt provisoire
→ Alt+clic (Windows) ou Option+clic (Macintosh) affiche la fenêtre des propriétés du point d'arrêt pour un point d'arrêt nouveau ou existant.
→ Une expression ou un objet sélectionné(e) peut être copié dans la zone d'évaluation par glisser-déposer.
→ Un clic sur du texte sélectionné tout en maintenant enfoncée la touche Ctrl (Windows) ou Commande (Mac OS) le copie dans la zone d'évaluation.
→ Ctrl+D (Windows) ou Commande+D (Mac OS) sur un texte sélectionné le copie dans la zone d'évaluation.

Toutes les sous-fenêtres

→ Ctrl+* (Windows) ou Commande+* (Mac OS) force la réactualisation des listes d'expressions.
→ Lorsqu'aucun objet n'est sélectionné dans les fenêtres, en appuyant sur **Entrée** vous avancez d'une ligne.
→ Lorsque la valeur d'un élément est sélectionnée, utilisez les touches directionnelles pour naviguer dans la liste.
→ Lorsque vous êtes en train d'éditer un élément, utilisez les touches directionnelles pour déplacer le curseur, utilisez Ctrl+A/X/C/V (Windows) ou Commande+A/X/C/V (Macintosh) en raccourci des commandes du menu **Edition** : Tout Sélectionner/Couper/Copier/Coller.

Référence

Débogueur, Fenêtre d'évaluation, Fenêtre d'évaluation des méthodes, Fenêtre d'expression, Fenêtre de chaîne d'appel.

11

Définition structure

Les commandes de ce thème retournent la description de la structure de la base. Elles permettent de connaître le nombre de tables, le nombre de champs dans chaque table, les noms des tables et des champs, ainsi que le type et les propriétés de chaque champ.

L'identification précise de la structure de la base est très utile quand vous développez et utilisez des groupes de méthodes projets et formulaires qui peuvent être copiées dans différentes bases.

La possibilité de lire la structure de la base vous permet de développer et d'utiliser du code portable.

Compter les tables et les champs
Depuis la version 11 de 4D, il est possible de supprimer des tables et des champs. Cette possibilité nécessite la modification des algorithmes utilisés dans les versions précédentes pour dénombrer les tables et les champs. Il est désormais nécessaire d'utiliser des algorithmes combinant les commandes Lire numero derniere table et Lire numero dernier champ et Est un numero de table valide et Est un numero de champ valide. Voici un exemple de ce type d'algorithme :

```
Boucle($latable;1;Lire numero derniere table)
   Si (Est un numero de table valide($latable))
      Boucle($lechamp;1;Lire numero dernier champ($latable))
         Si(Est un numero de champ valide($latable;$lechamp))
            ... `Le champ existe et est valide
         Fin de si
      Fin de boucle
   Fin de si
Fin de boucle
```

Référence

Champ, FIXER INDEX, Lire numero dernier champ, Lire numero derniere table, LIRE PROPRIETES CHAMP, Nom de la table, Pointeurs, Table.

Champ (numTable| ptrChamp{; numChamp}) → Num | Pointeur

Paramètre	Type		Description
numTable| ptrChamp	Num	Pointeur	→	Numéro de table ou Pointeur de champ
numChamp	Numérique	→	Numéro de champ si un numéro de table est passé
Résultat	Num	Pointeur	←	Numéro de champ si un pointeur de champ est passé, Pointeur de champ si des numéros de table et de champ sont passés

Description

La commande Champ a deux syntaxes :
• Si vous passez un numéro de table dans numTable et un numéro de champ dans numChamp, Champ retourne un pointeur vers le champ.
• Si vous passez un pointeur vers un champ dans ptrChamp, Champ retourne le numéro du champ.

Exemples

(1) L'exemple suivant assigne la variable ChampPtr à un pointeur vers le deuxième champ de la troisième table :

```
ChampPtr:=Champ(3; 2)
```

(2) Si vous passez champPtr (un pointeur vers le 2e champ de la table) à Champ, la valeur 2 est retournée. La ligne suivante assigne la valeur 2 à champNum :

```
champNum:=Champ(champPtr)
```

(3) Dans l'exemple, la variable champNum est égale au numéro de champ de [Table3]Champ2 :

```
champNum:=Champ(->[Table3]Champ2)
```

Référence

Lire numero dernier champ, LIRE PROPRIETES CHAMP, Nom du champ, Table.

CREER INDEX (laTable; tabChamps; typeIndex; nomIndex{; *})

Paramètre	Type		Description
laTable	Table	→	Table pour laquelle créer un index
tabChamps	Tableau pointeur	→	Pointeur(s) vers le(s) champ(s) à indexer
typeIndex	Entier	→	Type d'index à créer : -1 = Mots-clés, 0 = par défaut, 1 = B-Tree standard, 3 = BTree cluster
nomIndex	Texte	→	Nom de l'index à créer
*	*	→	Si passé = indexation asynchrone

Description

La commande CREER INDEX permet de créer :
• un index standard sur un ou plusieurs champs (index composite) ou
• un index de mots-clés sur un champ.

L'index est créé pour la table laTable en utilisant le ou les champ(s) désigné(s) par le tableau de pointeurs tabChamps. Ce tableau contient une seule ligne si vous souhaitez créer un index simple et deux ou plusieurs lignes si vous souhaitez créer un index composite (sauf index de mots-clés). Dans le cas d'index composites, l'ordre des champs dans le tableau est important lors de la construction de l'index.

Le paramètre typeIndex vous permet de définir le type d'index à créer. Vous pouvez passer une des constantes suivantes, placées dans le thème "Type index" :
• Index de mots clés (-1) : index de mots-clés. Attention, les index de mots-clés ne peuvent pas être composites : vous ne devez passer qu'un seul champ dans le tableau tabChamps.
• Type index par défaut (0) : dans ce cas, 4D définit le type d'index (hors index de mots-clés) le plus optimisé en fonction du contenu du champ.
• Index BTree standard (1) : index de type B-Tree classique. Ce type d'index polyvalent est utilisé dans les versions précédentes de 4D.
• Index BTree cluster (3) : index de type B-Tree utilisant des clusters. Ce type d'index est optimisé lorsque l'index contient peu de clés, c'est-à-dire lorsque les mêmes valeurs reviennent souvent dans les données.

Passez dans nomIndex le nom de l'index à créer. Ce nom est obligatoire, si vous passez une chaîne vide, une erreur est générée. Si l'index nomIndex existe déjà, la commande ne fait rien.

Le paramètre facultatif *, lorsqu'il est passé, permet d'effectuer l'indexation en mode asynchrone. Dans ce mode, la méthode d'origine poursuit son exécution après l'appel de la commande, que l'indexation soit terminée ou non.

Si la commande CREER INDEX rencontre des enregistrements verrouillés, elle ne les indexe pas et attend qu'ils soient libérés.

Si une erreur se produit durant l'exécution de la commande (champ non indexable, tentative de création d'index de mots-clés sur plusieurs champs, etc.), une erreur est générée. Cette erreur peut être interceptée à l'aide d'une méthode d'appel sur erreur.

Exemples

(1) Création de deux index standard sur les champs "Nom" et "Téléphone"de la table [Clients] :

```
TABLEAU POINTEUR(tabPtrChp;1)
tabPtrChp{1}:=->[Clients]Nom
CREER INDEX([Clients];tabPtrChp;Index BTree standard;"IdxCltNom")
tabPtrChp{1}:=->[Clients]Téléphone
CREER INDEX([Clients];tabPtrChp;Index BTree standard;"IdxCltTel")
```

(2) Création d'un index de mots-clés sur le champ "Observations" de la table [Clients] :

```
TABLEAU POINTEUR(tabPtrChp;1)
tabPtrChp{1}:=->[Clients]Observations
CREER INDEX([Clients];tabPtrChp;Index de mots clés;"IdxCltObs")
```

(3) Création d'un index composite sur les champs "CodePostal" et "Ville" de la table [Clients] :

```
TABLEAU POINTEUR(tabPtrChp;2)
tabPtrChp{1}:=->[Clients]CodePostal
tabPtrChp{2}:=->[Clients]Ville
CREER INDEX([Clients];tabPtrChp;Index BTree standard;"CPVille")
```

Référence

FIXER INDEX, SUPPRIMER INDEX.

Est un numero de champ valide

Est un numero de champ valide (numTable | ptrTable; numChamp) → Booléen

Paramètre	Type		Description
numTable | ptrTable	Num | Pointeur	→	Numéro de table ou Pointeur vers une table
numChamp	Entier long	→	Numéro de champ
Résultat	Booléen	←	Vrai = le champ existe dans la table, Faux = le champ n'existe pas dans la table

Description

La commande Est un numero de champ valide retourne Vrai si le champ dont le numéro est passé dans numChamp existe dans la table dont le numéro ou le pointeur est passé dans le paramètre numTable ou ptrTable. Si le champ n'existe pas, la commande retourne Faux. A noter que la commande retourne Faux si la table du champ se trouve dans la Corbeille de l'Explorateur.

Cette commande permet de détecter d'éventuelles suppressions de champs, ce qui crée des ruptures dans la séquence des numéros de champs.

Référence

Est un numero de table valide, Lire numero derniere table.

Est un numero de table valide (numTable) → Booléen

Paramètre	Type		Description
numTable	Entier long	→	Numéro de table
Résultat	Booléen	←	Vrai = la table existe dans la base, Faux = la table n'existe pas dans la base

Description

La commande Est un numero de table valide retourne Vrai si la table dont le numéro est passé dans numTable existe dans la base et Faux sinon. A noter que la commande retourne Faux si la table se trouve dans la Corbeille de l'Explorateur.

Cette commande permet de détecter d'éventuelles suppressions de tables, ce qui crée des ruptures dans la séquence des numéros de tables.

Référence

Est un numero de champ valide, Lire numero derniere table.

FIXER INDEX (champ; index{; mode}{; *})

Paramètre	Type		Description
champ	Champ	→	Champ duquel créer ou supprimer l'index
index	Booléen | Entier	→	• Vrai=Créer l'index, Faux=Supprimer l'index, ou • Créer un index de type : -1=mots-clés, 0=par défaut, 1=B-Tree standard, 3=B-Tree cluster
mode	Entier long	→	Obsolète (paramètre ignoré)
*		→	Indexation asynchrone si * est passé

Description

La commande FIXER INDEX admet deux syntaxes :
• Si vous passez un booléen dans le paramètre index, la commande crée ou supprime l'index du champ que vous avez passé dans champ.
• Si vous passez un entier dans le paramètre index, la commande crée un index du type spécifié.

index = booléen
Pour indexer le champ, passez Vrai dans index. La commande crée un index du type par défaut. Si l'index existe déjà, la commande ne fait rien.
Si vous passez Faux dans index, la commande supprimera tous les index non composites associés au champ. S'il n'existe pas d'index, la commande ne fait rien.

index = entier
Dans ce cas, la commande crée un index du type spécifié pour le champ. Vous pouvez passer une des constantes suivantes, placées dans le thème "Type index" :
• Index de mots clés (-1) : index de mots-clés, permettant l'indexation mot à mot du contenu du champ. Ce type d'index n'est utilisable qu'avec les champs de type Texte ou Alpha.
• Type index par défaut (0) : dans ce cas, 4D définit le type d'index (hors index de mots-clés) le plus optimisé en fonction du contenu du champ.
• Index BTree standard (1) : index de type B-Tree classique. Ce type d'index polyvalent est utilisé dans les versions précédentes de 4D.
• Index BTree cluster (3) : index de type B-Tree utilisant des clusters. Ce type d'index est optimisé lorsque l'index contient peu de clés, c'est-à-dire lorsque les mêmes valeurs reviennent souvent dans les données.

FIXER INDEX n'indexera pas les enregistrements verrouillés ; la commande attendra que les enregistrements soient libérés.

Depuis la version 11, le paramètre mode est inutile et est ignoré s'il est passé.

Le paramètre optionnel * indique une indexation asynchrone (simultanée). Une indexation asynchrone permet à la méthode appelante de poursuivre son exécution immédiatement après l'appel, que l'indexation soit terminée ou non. Cependant, l'exécution sera stoppée si une commande requiert l'index.

Notes :
• Les index créés par cette commande ne portent pas de nom. Ils ne pourront pas être supprimés par la commande SUPPRIMER INDEX via la syntaxe basée sur le nom.
• Cette commande ne permet pas de créer ou de supprimer des index composites.
• Cette commande ne permet pas de supprimer un index de mots-clés créé par la commande CREER INDEX.

Exemples

(1) L'exemple suivant indexe le champ [Clients]Num :

```
LIBERER ENREGISTREMENT([Clients])
FIXER INDEX ([Clients]Num; Vrai)
```

(2) Vous souhaitez indexer le champ [Clients]Nom, en mode asynchrone :

```
FIXER INDEX([Clients]Nom;Vrai;*)
```

(3) Création d'un index de mots-clés :

```
FIXER INDEX([Livres]Résumé;Index de mots clés)
```

Référence

CHERCHER, CREER INDEX, LIRE PROPRIETES CHAMP, SUPPRIMER INDEX, TRIER.

LIRE PROPRIETES CHAMP

Définition structure

version 11 (Modifiée)

LIRE PROPRIETES CHAMP (chpPtr | tableNum{; champNum}; champType{; champLong{; indexé{; unique{; invisible}}}})

Paramètre	Type		Description
chpPtr | tableNum	Pointeur | Num	→	Pointeur de champ ou Numéro de table
champNum	Numérique	→	Numéro de champ si un numéro de table est passé en premier paramètre
champType	Numérique	←	Type de champ
champLong	Numérique	←	Longueur du champ (si alphanumérique)
indexé	Booléen	←	Vrai = Indexé, Faux = Non indexé
unique	Booléen	←	Vrai = Unique, Faux = Non unique
invisible	Booléen	←	Vrai = Invisible, Faux = Visible

Description

La commande LIRE PROPRIETES CHAMP retourne des informations sur le champ désigné par tableNum et champNum ou par chpPtr.

Vous pouvez soit passer :
• les numéros de table et de champ dans tableNum et champNum
• ou un pointeur vers le champ dans chpPtr.

Après l'appel :

• Le paramètre champType retourne le type du champ. Le paramètre variable champType reçoit l'une des valeurs prédéfinies par les constantes de 4D :

Constante	Type	Valeur
Est un champ alpha	Entier long	0
Est un texte	Entier long	2
Est un numérique	Entier long	1
Est un float	Entier long	35
Est un entier	Entier long	8
Est un entier long	Entier long	9
Est un entier 64 bits	Entier long	25
Est une date	Entier long	4

Est une heure	Entier long	11
Est un booléen	Entier long	6
Est une image	Entier long	3
Est une sous table	Entier long	7
Est un BLOB	Entier long	30

• Le paramètre champLong retourne la longueur du champ si celui-ci est de type Alpha (ce qui signifie que vous obtenez champType=Est un champ alpha). La valeur de champLong n'est pas significative pour les autres types de champ.

• Le paramètre indexé retourne Vrai si le champ est indexé, Faux sinon. La valeur de indexé est significative pour les champs de type Alphanumérique, Entier, Entier long, Réel, Date, Heure et Booléen.

• Le paramètre unique retourne Vrai si le champ dispose de l'attribut "Unique", Faux sinon. L'attribut Unique ne peut être appliqué qu'aux champs indexés.

• Le paramètre invisible retourne Vrai si le champ dispose de l'attribut "Invisible", Faux sinon. L'attribut Invisible permet de masquer le champ dans les éditeurs standard de 4D (étiquettes, graphes...).

Exemples

(1) Dans l'exemple suivant, les variables vType, vLong, vIndex, vUnique et vInvisible prennent pour valeur les propriétés du troisième champ de la première table :

```
LIRE PROPRIETES CHAMP(1; 3;vType;vLong;vIndex;vUnique;vInvisible)
```

(2) L'exemple suivant récupère dans les variables vType, vLong, vIndex, vUnique et vInvisible les propriétés du champ [Table3]Champ2 :

```
LIRE PROPRIETES CHAMP(->[Table3]Champ2;vType;vLong;vIndex;vUnique;vInvisible)
```

Référence

Champ, FIXER INDEX, Nom du champ.

LIRE PROPRIETES LIEN (ptrChp|numTable{; numChamp}; tableDest; champDest{; discriminant{; allerAuto{; retourAuto}}})

Paramètre	Type		Description
ptrChp|numTable	Pointeur | Entier long	→	Pointeur de champ ou Numéro de table
numChamp	Entier long	→	Numéro de champ si un numéro de table est passé en premier paramètre
tableDest	Entier long	←	Numéro de la table cible ou 0 si aucun lien ne part du champ
champDest	Entier long	←	Numéro du champ cible ou 0 si aucun lien ne part du champ
discriminant	Entier long	←	Numéro du champ discriminant ou 0 si aucun champ discriminant
allerAuto	Booléen	←	Vrai = Lien aller automatique, Faux = Lien aller manuel
retourAuto	Booléen	←	Vrai = Lien retour automatique, Faux = Lien retour manuel

Description

La commande LIRE PROPRIETES LIEN retourne les propriétés du lien, s'il y en a un, qui part du champ source, désigné par numTable et numChamp ou par ptrChp.
Vous pouvez passer :
• soit des numéros de table et de champ dans numTable et numChamp,
• soit un pointeur vers le champ dans ptrChp.

Après l'exécution de la commande :
• Les paramètres tableDest et champDest contiennent respectivement le numéro de la table et du champ vers lesquels pointe le lien partant du champ source. Si aucun lien ne part du champ, ces paramètres contiennent 0.
• Le paramètre discriminant contient le numéro du champ discriminant (appartenant à la table cible) défini pour le lien. Si aucun champ discriminant n'a été défini pour le lien ou si aucun lien ne part du champ source, ce paramètre contient 0.
• Les paramètres allerAuto et retourAuto retournent Vrai si respectivement les options "Lien aller auto" et "Lien retour auto" ont été cochées pour le lien, Faux sinon.

Note : Les deux derniers paramètres retournent également Vrai si aucun lien ne part du champ source (dans ce cas, leur valeur est non significative). La valeur des paramètres tableDest et champDest vous permet de vous assurer de l'existence d'un lien.

Référence

FIXER LIEN CHAMP, FIXER LIENS AUTOMATIQUES, LIRE PROPRIETES CHAMP, LIRE PROPRIETES SAISIE CHAMP, LIRE PROPRIETES TABLE.

LIRE PROPRIETES SAISIE CHAMP (ptrChp|numTable{; numChamp}; nomEnum; obligatoire; nonSaisissable; nonModifiable)

Paramètre	Type		Description
ptrChp|numTable	Pointeur|Entier long	→	Pointeur de champ ou Numéro de table
numChamp	Entier long	→	Numéro de champ si un numéro de table est passé en premier paramètre
nomEnum	Alpha	←	Nom de l'énumération associée ou Chaîne vide
obligatoire	Booléen	←	Vrai = Obligatoire, Faux = Facultatif
nonSaisissable	Booléen	←	Vrai = Non saisissable, Faux = Saisissable
nonModifiable	Booléen	←	Vrai = Non modifiable, Faux = Modifiable

Description

La commande LIRE PROPRIETES SAISIE CHAMP retourne les propriétés relatives à la saisie de données du champ désigné par numTable et numChamp ou par ptrChp.
Vous pouvez passer :
• soit des numéros de table et de champ dans numTable et numChamp,
• soit un pointeur vers le champ dans ptrChp.

Les propriétés retournées par cette commande sont celles qui ont été définies au niveau de la fenêtre de structure de la base. Des propriétés similaires peuvent également être définies au niveau des formulaires.

Après l'exécution de la commande :
• Le paramètre nomEnum contient le nom de l'énumération associée au champ, s'il y en a une. Il est possible d'associer un énumération aux champs de type Alpha, Texte, Numérique, Entier, Entier long, Date, Heure et Booléen.
Si aucune énumération n'est associée au champ, ou si son type n'admet pas l'association d'énumération, une chaîne vide ("") est retournée.
• Le paramètre obligatoire retourne Vrai si le champ dispose de l'attribut "Obligatoire", Faux sinon. L'attribut "Obligatoire" peut être associé aux champs de tous types, hormis sous-table et BLOB.
• Le paramètre nonSaisissable retourne Vrai si le champ dispose de l'attribut "Non saisissable", Faux sinon. Un champ non saisissable ne peut qu'être lu, il n'accepte aucune saisie de données. L'attribut "Non saisissable" peut être associé aux champs de tous types, hormis sous-table et BLOB.

• Le paramètre nonModifiable retourne Vrai si le champ dispose de l'attribut "Non modifiable", Faux sinon. Un champ non modifiable n'accepte qu'une seule saisie, et ne peut plus être modifié par la suite. L'attribut "Non modifiable" peut être associé aux champs de tous types, hormis sous-table et BLOB.

Référence

LIRE PROPRIETES CHAMP, LIRE PROPRIETES LIEN, LIRE PROPRIETES TABLE.

LIRE PROPRIETES TABLE (ptrTable|numTable; invisible{; trigSvgdeNouv{; trigSvgdeEnr{; trigSupprEnr{; trigChargEnr}}}})

Paramètre	Type		Description
ptrTable|numTable	Pointeur|Entier long	→	Pointeur de table ou Numéro de table
invisible	Booléen	←	Vrai = Invisible, Faux = Visible
trigSvgdeNouv	Booléen	←	Vrai = Trigger “Sur sauvegarde nouvel enreg” activé, sinon Faux
trigSvgdeEnr	Booléen	←	Vrai = Trigger “Sur sauvegarde enregistrement” activé, sinon Faux
trigSupprEnr	Booléen	←	Vrai = Trigger “Sur suppression enreg” activé, sinon Faux
trigChargEnr	Booléen	←	*** Ne pas utiliser (obsolète) ***

Description

La commande LIRE PROPRIETES TABLE retourne les propriétés de la table désignée par ptrTable ou numTable. Vous pouvez passer dans le premier paramètre soit un pointeur vers la table, soit le numéro de la table.

Après l’exécution de la commande :
• Le paramètre invisible retourne Vrai si la table dispose de l’attribut “Invisible”, Faux sinon. L’attribut “Invisible” permet de masquer la table dans les éditeurs standard de 4D (étiquettes, graphes...).
• Le paramètre trigSvgdeNouv retourne Vrai si le trigger “Sur sauvegarde nouvel enreg” a été activé pour la table, Faux sinon.
• Le paramètre trigSvgdeEnr retourne Vrai si le trigger “Sur sauvegarde enregistrement” a été activé pour la table, Faux sinon.
• Le paramètre trigSupprEnr retourne Vrai si le trigger “Sur suppression enreg” a été activé pour la table, Faux sinon.

Référence

LIRE PROPRIETES CHAMP, LIRE PROPRIETES LIEN, LIRE PROPRIETES SAISIE CHAMP.

Nom de la table (numTable | ptrTable) → Alpha

Paramètre	Type		Description
numTable | ptrTable	Num | Pointeur	→	Numéro de table ou pointeur de table
Résultat	Alpha	←	Nom de la table

Description

Nom de la table retourne le nom de la table dont le numéro ou le pointeur a été passé dans numTable ou ptrTable.

Exemple

La méthode suivante est un exemple de méthode générique qui affiche les enregistrements d'une table. La référence à la table est passée en tant que pointeur vers la table. La commande Nom de la table est utilisée pour inclure le nom de la table dans la barre de titre de la fenêtre :

```
` Méthode projet AFFICHER SELECTION COURANTE
` AFFICHER SELECTION COURANTE (Pointeur)
` AFFICHER SELECTION COURANTE (->[Table] )

` Fixer le titre de la fenêtre
CHANGER TITRE FENETRE(" Enregistrements pour "+Nom de la table($1))
VISUALISER SELECTION($1>>) ` Afficher la sélection
```

Référence

Nom du champ, Nombre de tables, Table.

Nom du champ (champPtr | tableNum {; champNum}) → Alpha

Paramètre	Type		Description
champPtr | tableNum	Numérique	→	Pointeur vers un champ ou Numéro de table
champNum	Numérique	→	Numéro de champ si un numéro de table est passé en premier paramètre
Résultat	Alpha	←	Nom du champ

Description

La commande Nom du champ retourne le nom du champ dont vous avez passé le pointeur dans champPtr, ou dont vous avez passé les numéros de table et de champ dans tableNum et champNum.

Exemples

(1) L'exemple suivant assigne au second élément du tableau ChampTableau{1} (ChampTableau étant un tableau à deux dimensions) le nom du second champ de la première table :

```
ChampTableau{1}{2}:=Nom du champ(1;2)
```

(2) L'exemple suivant assigne au second élément du tableau ChampTableau{1} (ChampTableau étant un tableau à deux dimensions) le nom du champ [MaTable]MonChamp :

```
ChampTableau{1}{2}:=Nom du champ(->[MaTable]MonChamp)
```

(3) L'exemple suivant affiche une boîte de dialogue d'alerte. Nous passons à cette méthode un pointeur vers un champ :

```
ALERTE("Le numéro du champ "+Nom du champ($1)+" de la table "
                    +Nom de la table(Table($1))+" doit faire plus de cinq caractères.")
```

Référence

Champ, Nom de la table, Nombre de champs.

Lire numero dernier champ

Lire numero dernier champ (numTable | ptrTable) → Numérique

Paramètre	Type		Description
numTable | ptrTable	Num | Pointeur	→	Numéro de table ou Pointeur vers une table
Résultat	Numérique	←	Numéro de champ le plus élevé dans la table

Description

La commande Lire numero dernier champ retourne le numéro de champ le plus élevé parmi les champs de la table dont le numéro ou le pointeur est passé dans le paramètre numTable ou ptrTable.

Les champs sont numérotés dans l'ordre où ils ont été créés. Si aucun champ n'a été supprimé dans la table, cette commande retourne donc le nombre de champs que contient la table. Dans le cadre de boucles itératives sur les numéros de champs de la table, vous devez utiliser la commande Est un numero de champ valide afin de vérifier que le champ n'a pas été supprimé.

Exemple

La méthode projet suivante crée le tableau taChamps avec les noms des champs de la table dont le pointeur est reçu en paramètre :

```
$vlTable:=Table($1)
TABLEAU ALPHA(31;taChamps;Lire numero dernier champ($vlTable))
Boucle ($vlChamp;Taille tableau(taChamps);1;-1)
    Si(Est un numero de champ valide($vlTable;$vlChamp))
        taChamps{$vlChamp}:=Nom du champ($vlTable;$vlChamp)
    Sinon
        SUPPRIMER DANS TABLEAU(taChamps; $vlChamp)
    Fin de si
Fin de boucle
```

Référence

Est un numero de champ valide, Lire numero derniere table, LIRE PROPRIETES CHAMP, Nom du champ.

Lire numero derniere table

Lire numero derniere table → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Numéro de table le plus élevé dans la base

Description

Lire numero derniere table retourne le numéro de table le plus élevé parmi les tables de la base. Les tables sont numérotées dans l'ordre dans lequel elles ont été créées. Si aucune table n'a été supprimée dans la base, cette commande retourne donc le nombre de tables présentes dans la base. Dans le cadre de boucles itératives sur les numéros de tables de la base, vous devez utiliser la commande Est un numero de table valide afin de vérifier que la table n'a pas été supprimée.

Exemple

L'exemple suivant initialise les éléments du tableau *tabTables*. Ce tableau peut être utilisé comme liste déroulante (ou onglets, zone de défilement, etc.) pour afficher dans un formulaire la liste des tables de la base :

```
TABLEAU ALPHA (31; tabTables; Lire numero derniere table)
Si (Lire numero derniere table>0)  `si la base contient bien des tables
   Boucle ($vlTables;Taille tableau(tabTables);1;-1)
      Si(Est un numero de table valide($vlTables))
         tabTables{$vlTables}:= Nom de la table ($vlTables)
      Sinon
         SUPPRIMER DANS TABLEAU(tabTables; $vlTables)
      Fin de si
   Fin de boucle
Fin de si
```

Référence

Est un numero de table valide, Lire numero dernier champ, Nom de la table.

SUPPRIMER INDEX (ptrChp | nomIndex{; *})

Paramètre	Type		Description
ptrChp | nomIndex	Pointeur | Texte	→	Pointeur vers le champ duquel supprimer les index ou Nom de l'index à supprimer
*	*	→	Si passé = opération asynchrone

Description

La commande SUPPRIMER INDEX permet de supprimer un ou plusieurs index existant dans la base. Vous pouvez passer en paramètre soit un pointeur vers un champ, soit un nom d'index :
• Si vous passez un pointeur vers un champ (ptrChp), tous les index associés au champ seront supprimés. Il peut s'agir d'index de mots-clés ou d'index standard. Si le champ est inclus dans un ou plusieurs index composite(s), ils sont également supprimés.
• Si vous passez nom d'index (nomIndex), seul l'index désigné sera supprimé. Il peut s'agir d'index de mots-clés ou d'index standard. Si le champ comporte d'autres index ou appartient à d'autres index composites, ils ne sont pas supprimés.

Le paramètre facultatif *, lorsqu'il est passé, permet d'effectuer la désindexation en mode asynchrone. Dans ce mode, la méthode d'origine poursuit son exécution après l'appel de la commande, que la suppression d'index soit terminée ou non.

S'il n'existe pas d'index correspondant à ptrChp ou à nomIndex, la commande ne fait rien.

Exemple

Cet exemple illustre les deux syntaxes de la commande :

```
    `Suppression de tous les index liés au champ Nom
SUPPRIMER INDEX(->[Clients]Nom)
    `Suppression de l'index nommé "CPVille"
SUPPRIMER INDEX("CPVille")    `
```

Référence

CREER INDEX, FIXER INDEX.

Table (numTable | unPtr) → Num | Pointeur

Paramètre	Type		Description
numTable | unPtr	Num | Pointeur	→	Numéro de table ou Pointeur de table ou Pointeur de champ
Résultat	Num | Pointeur	←	Pointeur de table si un Numéro de table est passé, Numéro de table si un Pointeur de table est passé, Numéro de table si un Pointeur de champ est passé

Description

Table a trois syntaxes différentes.
• Si vous passez un numéro de table dans numTable, Table retourne un pointeur sur la table.
• Si vous passez un pointeur de table dans unPtr, Table retourne le numéro de la table.
• Si vous passez un pointeur de champ dans unPtr, Table retourne le numéro de table du champ.

Exemples

(1) Dans cet exemple, la variable ptrTable reçoit un pointeur sur la table n°3 :

```
ptrTable := Table (3)
```

(2) Si vous passez ptrTable à la fonction Table, elle retourne 3. Par exemple, dans la ligne suivante, la variable numTable prend la valeur 3 :

```
numTable := Table (ptrTable)
```

(3) Dans l'exemple suivant, la variable numTable est égale au numéro de la table [Table3] :

```
numTable := Table (->[Table3])
```

(4) Dans l'exemple suivant, la variable numTable est égale au numéro de la table à laquelle appartient le champ [Table3]Champ1 :

```
numTable := Table (->[Table3]Champ1)
```

Référence

Champ, Nom de la table, Nombre de tables, Pointeurs.

12

Documents système

Introduction

Tous les documents et applications que vous utilisez sur votre ordinateur sont stockés en tant que **fichiers** sur le ou les disques durs **connectés** ou **montés** sur votre ordinateur, ou encore sur des disquettes et autres supports de stockage permanent. Dans cette documentation ainsi que dans 4D, les termes **fichier** ou **document** sont indifféremment employés pour désigner ces documents et applications. Cependant, la plupart des commandes de ce thème utilisent le mot document car, généralement, vous les utiliserez pour accéder à des documents (par opposition à des fichiers d'application ou des fichiers système) sur disque.

Un disque dur peut être formaté de manière à comporter une ou plusieurs partitions. Chaque partition s'appelle un **volume**. Peu importe que ces volumes soient des partitions physiquement présentes sur le même disque dur ou non, au niveau de 4D, ces volumes sont considérés comme des entités séparées et équivalentes.

Un volume peut être situé sur un disque dur physiquement connecté à votre machine ou monté par le réseau par l'intermédiaire d'un protocole de partage de fichiers tel que NetBEUI (Windows) ou AFP (Macintosh). Quel que soit le cas, au niveau de 4D, ces volumes sont considérés de la même façon lorsque vous utilisez les commandes du thème Documents Système (à moins que vous n'en décidiez autrement et utilisiez des plug-ins 4D pour étendre les capacités de votre application dans ce domaine).

Chaque volume a un **nom de volume**. Sous Windows, les volumes sont désignés par une lettre suivie de deux points. Généralement, A: et B: sont utilisés pour désigner les lecteurs de disquettes et C: désigne le volume que vous utilisez pour lancer votre système (à moins que vous n'ayez configuré votre PC différemment). Ensuite, les lettres D: à Z: sont utilisées pour les volumes supplémentaires connectés à votre PC (lecteurs DVD, autres lecteurs, lecteurs réseau, etc.). Sous Mac OS, les volumes ont des noms communs (ces noms sont ceux que vous visualisez sur le bureau au niveau du Finder).

Normalement, vous classez vos documents dans des **dossiers** qui peuvent eux-mêmes contenir d'autres dossiers. Il n'est pas conseillé d'accumuler des centaines ou des milliers de fichiers au même niveau d'un volume. C'est un fouillis, qui de plus qui ralentit votre système. Sous Windows, un dossier est parfois encore appelé un “répertoire”.

Pour identifier un document de manière certaine, vous avez besoin de connaître le nom du volume, le nom du ou des dossiers(s) dans le(s)quel(s) se trouve le document, ainsi que le nom du document lui-même.

Si vous concaténez tous ces noms, vous obtenez le **chemin d'accès** à ce document. Dans le nom de ce chemin, les noms de dossiers sont séparés par un caractère spécial appelé **symbole séparateur** (de répertoire). Sous Windows, ce caractère est la barre oblique inversée \, sous Mac OS ce sont les deux-points :

Examinons un exemple. Vous disposez d'un document Important situé dans le dossier Mémos, lui-même situé dans le dossier Documents, lui-même situé dans le dossier EnCours.

Si, sous Windows, l'ensemble est situé sur le volume C: , le chemin d'accès au document est donc :
C:\EnCours\Documents\Mémos\Important.TXT

Note : Le caractère \ est également utilisé par l'éditeur de méthodes de 4D pour désigner des séquences d'échappement. Pour éviter tout problème d'interprétation, l'éditeur transforme automatiquement les chemins d'accès du type C:\Disque en C:\\Disque. Pour plus d'informations, reportez-vous ci-dessous au paragraphe “Spécification des noms et chemins d'accès des documents”.

Si, sous Mac OS, l'ensemble est situé sur le volume Interne, le chemin d'accès au document est donc :
Interne:EnCours:Documents:Mémos:Important

Notez que, sous Windows, avec cet exemple, le nom du document contient le suffixe .TXT. Nous verrons pourquoi plus loin dans cette section.

Quelle que soit la plate-forme, le chemin d'accès à un document peut être exprimé sous la forme suivante : VolNom DosSep { DosNom DosSep { DosNom DosSep { ... } } } DocNom.

Tous les documents (fichiers) situés sur des volumes ont plusieurs caractéristiques généralement appelées **attributs** ou **propriétés** : par exemple le **nom** du document lui-même.

Type de document et Créateur

Sous Windows, un document a un **type**. Sous Mac OS, un document a également un **type** et peut en outre avoir un **créateur** (ou "creator"). Le type d'un document indique généralement ce qu'est le document ou ce qu'il contient. Par exemple, un document de type texte contient du texte (sans style). Le type d'un document est déterminé par son suffixe, appelé **extension de fichier**, rattaché au nom du document. Par exemple .TXT ou .TEXT est l'extension de fichier pour des documents texte.

Ce principe est identique sous Mac OS X, toutefois par compatibilité avec les versions précédentes du Système, le type d'un document peut être déterminé par la propriété **type de fichier** si elle a été définie. Cette propriété est une signature sur 4 caractères (non affichée au niveau du Finder). Par exemple, le type de fichier d'un document texte est "TEXT". De plus, un document peut être identifié par un créateur, désignant l'application qui a créé le document. Cette notion n'existe pas sous Windows. Le créateur d'un document est déterminé par la propriété de **créateur de fichier** du document. Si un document dispose de propriétés type et créateur, Mac OS les prendra en compte quelle que soit l'extension de ce document.

DocRef : numéro de référence de document

Un document est **ouvert en mode lecture/écriture**, **ouvert en mode lecture** ou **fermé**. Avec les commandes 4D, un document ne peut être ouvert en mode lecture/écriture que par un process à la fois. Un process peut ouvrir plusieurs documents, plusieurs process peuvent ouvrir de multiples documents, vous pouvez ouvrir un même document en mode lecture autant de fois que nécessaire, mais vous ne pouvez pas ouvrir le même document en mode lecture/écriture deux fois en même temps.

Vous ouvrez un document à l'aide des fonctions Ouvrir document, Creer document et Ajouter a document. Les fonctions Creer document et Ajouter a document ouvrent automatiquement les documents en mode lecture/écriture. Seule la fonction Ouvrir document permet de choisir le mode d'ouverture.
Une fois que le document est ouvert en lecture/écriture, vous pouvez lire et écrire des caractères dans ce document (cf. les commandes RECEVOIR PAQUET et ENVOYER PAQUET). Lorsque vous en avez terminé avec un document, il est préférable de le fermer — avec la commande FERMER DOCUMENT.

Tous les documents ouverts sont désignés au moyen de l'expression **DocRef**, retournée par les commandes Ouvrir document, Creer document et Ajouter a document. Une DocRef identifie de façon unique un document ouvert. C'est une expression de type Heure. Toutes les commandes fonctionnant avec des documents ouverts attendent une DocRef comme paramètre. Si vous passez une DocRef incorrecte à l'une de ces commandes, une erreur du gestionnaire de fichiers est générée.

Gestion des erreurs E/S

Quand vous accédez à des documents (ouverture, fermeture, suppression, changement de nom, copie), quand vous modifiez les propriétés d'un document ou quand vous lisez et écrivez des caractères dans un document, des erreurs d'entrée/sortie (E/S) peuvent se produire. Un document peut ne pas avoir été trouvé ; il peut être verrouillé ; il peut être déjà ouvert en

écriture. Vous pouvez repérer ces erreurs grâce à une méthode de gestion des erreurs installée par la commande APPELER SUR ERREUR. La plupart des erreurs qui peuvent se produire lors de l'utilisation des commandes du thème documents système est décrite dans la section Erreurs du gestionnaire de fichiers du système.

La variable système Document

Les commandes Ouvrir document, Creer document, Ajouter a document et Selectionner document vous permettent d'accéder à un document par les boîtes de dialogue standard d'ouverture ou d'enregistrement de fichiers. Quand vous accédez à un document par ces boîtes de dialogue standard, 4D retourne le chemin d'accès complet du document dans la variable système Document. Ne confondez pas cette variable système avec le paramètre document qui apparaît dans la liste des paramètres des commandes.

Spécification des noms et chemins d'accès des documents

La plupart des routines de cette section attendant un nom de document acceptent à la fois un nom et un chemin d'accès au document (*). Si vous passez un nom, la commande cherche le document dans le dossier de la base. Si vous passez un chemin d'accès, il doit être valide.

Si vous passez un nom ou un chemin d'accès incorrect, la commande génère une erreur du gestionnaire de fichiers que vous pouvez intercepter avec une méthode d'APPELER SUR ERREUR.

(*) sauf cas contraire spécifié explicitement.

Saisie de chemins d'accès Windows et séquences d'échappement
L'éditeur de méthodes de 4D permet d'utiliser des séquences d'échappement. Une séquence d'échappement est une suite de caractères permettant de remplacer un caractère "spécial". La séquence débute par le caractère barre oblique inversée (antislash) \, suivi d'un caractère. Par exemple, \t est une séquence d'échappement pour le caractère Tabulation.
Le caractère \ est aussi utilisé comme séparateur dans les chemins d'accès sous Windows. En général, 4D interprétera correctement les chemins d'accès Windows saisis dans l'éditeur de méthodes en remplaçant automatiquement les barres simples \ par des doubles barres \\. Par exemple, C:\Dossier deviendra C:\\Dossier.
Toutefois, si vous écrivez C:\MesDocuments\Nouveaux, 4D affichera C:\\MesDocuments\Nouveaux. Dans ce cas, le second \ est incorrectement interprété \N (séquence d'échappement existante). Vous devez donc saisir un double \\ lorsque vous souhaitez insérer une barre oblique inversée devant un caractère utilisé dans une des séquences d'échappement reconnues par 4D.

Les séquences d'échappement reconnues par 4D sont les suivantes :

Séquence d'échappement	Caractère remplacé
\n	LF (Retour ligne)
\t	HT (Tabulation)
\r	CR (Retour chariot)
\\	\ (Barre oblique inversée)
\"	" (Guillemets)

Méthodes projet utiles pour la gestion des documents sur disque

• Détecter sur quelle plate-forme vous opérez

Bien que 4D fournisse des commandes telles que ASSOCIER TYPES FICHIER destinées à éliminer les modifications de code liées aux particularités des plates-formes, lorsque vous commencez à travailler à un plus bas niveau en manipulant des documents sur disque, par exemple lorsque vous obtenez les chemins d'accès par programmation, vous avez besoin de savoir si vous fonctionnez sous Windows ou Mac OS.

La méthode projet Sous Windows listée ci-dessous vous permet de savoir si votre base tourne sous Windows :

```
` Méthode projet Sous Windows
` Sous Windows -> Booléen
` Sous Windows -> Vrai si la base est sous Windows

C_BOOLEEN($0)
C_ENTIER LONG($vlPlatform;$vlSystem;$vlMachine)

PROPRIETES PLATE FORME($vlPlatform;$vlSystem;$vlMachine)
$0:=($vlPlatform=Windows)
```

• Utiliser le bon symbole séparateur de dossiers

Sous Windows, un niveau de dossier est symbolisé par une barre oblique inversée \. Sous Mac OS, un niveau de dossier est symbolisé par deux-points :
En fonction de la plate-forme sur laquelle tourne la base, la méthode projet Symbole séparateur suivante vous retourne le code du caractère séparateur de dossiers.

```
` Méthode projet Symbole séparateur
` Symbole séparateur -> Entier
    ` Symbole séparateur -> code de "\\" (Windows) ou ":" (Mac OS)
```

```
C_ENTIER($0)

Si (Sous Windows )
    $0:=Code de caractere("\\")
Sinon
    $0:=Code de caractere(":")
Fin de si
```

• Extraire le nom de fichier d'un chemin d'accès complet (ou "nom long")

Une fois que vous avez récupéré le "nom long" d'un fichier (c'est-à-dire le chemin d'accès+le nom du fichier), vous pouvez avoir besoin d'en extraire le nom du fichier seul, par exemple pour l'afficher comme titre d'une fenêtre. La méthode projet Nom long vers nom de fichier vous le permet, sous Windows et Mac OS.

```
` Méthode projet Nom long vers nom de fichier
` Nom long vers nom de fichier ( Chaîne ) -> Chaîne
` Nom long vers nom de fichier ( nom long ) -> nom de fichier

C_ALPHA(255;$1;$0)
C_ENTIER($viLen;$viPos;$viChar;$viDirSymbol)

$viDirSymbol:=Symbole séparateur
$viLen:=Longueur($1)
$viPos:=0
Boucle ($viChar;$viLen;1;-1)
    Si (Code de caractere($1[[$viChar]])=$viDirSymbol)
        $viPos:=$viChar
        $viChar:=0
    Fin de si
Fin de boucle
Si ($viPos>0)
    $0:=Sous chaine($1;$viPos+1)
Sinon
    $0:=$1
Fin de si
Si (<>vbDebugOn) ` Mettre la variable à Vrai ou Faux dans la méthode base Sur ouverture
    Si ($0="")
        TRACE
    Fin de si
Fin de si
```

• Extraire le chemin d'accès seul du chemin d'accès complet (ou "nom long")

Une fois que vous avez récupéré le "nom long" d'un fichier (c'est-à-dire le chemin d'accès+le nom du fichier), vous pouvez avoir besoin d'en extraire uniquement le chemin d'accès au fichier, par exemple pour sauvegarder d'autres documents au même endroit. La méthode projet Nom long vers chemin accès vous le permet, sous Windows et Mac OS.

```
` Méthode projet Nom long vers chemin accès
` Nom long vers chemin accès ( Chaîne ) -> Chaîne
` Nom long vers chemin accès ( nom long ) -> chemin d'accès

C_ALPHA(255;$1;$0)
C_ALPHA(1;$vsDirSymbol)
C_ENTIER($viLen;$viPos;$viChar;$viDirSymbol)
$viDirSymbol:=Symbole séparateur
$viLen:=Longueur($1)
$viPos:=0
Boucle ($viChar;$viLen;1;-1)
   Si (Code de caractere($1[[$viChar]])=$viDirSymbol)
      $viPos:=$viChar
      $viChar:=0
   Fin de si
Fin de boucle
Si ($viPos>0)
   $0:=Sous chaine($1;1;$viPos)
Sinon
   $0:=$1
Fin de si
Si (<>vbDebugOn) ` Mettre la variable à Vrai ou Faux dans la méthode base Sur ouverture
   Si ($0="")
      TRACE
   Fin de si
Fin de si
```

Référence

Ajouter a document, ASSOCIER TYPES FICHIER, CHANGER CREATEUR DOCUMENT, CHANGER POSITION DANS DOCUMENT, CHANGER PROPRIETES DOCUMENT, CHANGER TAILLE DOCUMENT, CHANGER TYPE DOCUMENT, COPIER DOCUMENT, Createur document, Creer document, CREER DOSSIER, DEPLACER DOCUMENT, FERMER DOCUMENT, LISTE DES DOCUMENTS, LISTE DES DOSSIERS, LISTE DES VOLUMES, Ouvrir document, Position dans document, PROPRIETES DOCUMENT, PROPRIETES DU VOLUME, Selectionner document, SUPPRIMER DOCUMENT, Taille document, Tester chemin acces, Type document.

Ajouter a document (document{; type}) → DocRef

Paramètre	Type		Description
document	Alpha	→	Nom du document ou Chemin d'accès complet au document ou Chaîne vide pour afficher la boîte de dialogue standard d'ouverture de fichiers
type	Alpha	→	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
Résultat	DocRef	←	Numéro de référence du document

Description

La commande Ajouter a document "fait la même chose" que la commande Ouvrir document : elle vous permet d'ouvrir un document sur disque.

La seule différence est que Ajouter a document se place initialement à la fin du document, alors que Ouvrir document se place au début.

Pour plus d'informations, reportez-vous à la description de la commande Ouvrir document.

Exemple

L'exemple suivant ouvre un document existant qui s'appelle “Note”, ajoute à la fin du document la chaîne “ et à bientôt” suivie d'un retour chariot puis le referme. Si le document contenait déjà la chaîne “Au revoir”, il contiendra la chaîne “Au revoir et à bientôt” suivie d'un retour chariot :

```
C_HEURE(vDoc)
vDoc := Ajouter a document ("Note.txt")  ` Ouvrir le document Note
ENVOYER PAQUET (vDoc; " et à bientôt" + Caractere (13))  ` Ajouter la chaîne
FERMER DOCUMENT (vDoc)  ` Fermer le document
```

Variables et ensembles système

Si le document est correctement ouvert, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Après l'appel, la variable système Document contient le nom complet du document.

Référence

Creer document, Ouvrir document.

ASSOCIER TYPES FICHIER (macOS; windows; contexte)

Paramètre	Type		Description
macOS	Alpha	→	Type de fichier Mac OS (4 caractères)
windows	Alpha	→	Extension de fichier Windows
contexte	Alpha	→	Chaîne affichée dans la liste déroulante des types de fichiers dans la boîte de dialogue d'ouverture de fichiers sous Windows

Description

ASSOCIER TYPES FICHIER vous permet d'associer une extension de fichier Windows à un type de fichier Mac OS.

Il suffit d'appeler cette routine une fois seulement pour associer des types de fichier dans une session de travail entière avec une base. Une fois que vous l'avez appelée, les commandes Ajouter a document, Creer document, Creer fichier ressources et Ouvrir fichier ressources, lorsqu'elle sont exécutées sous Windows, vont automatiquement substituer l'extension de fichier Windows au type de fichier Mac OS que vous avez passé en paramètre à cette routine.

Dans le paramètre macOS, passez un type de fichier Macintosh de 4 caractères. Si vous ne passez pas une chaîne valide représentant un type de fichier, la commande ne fait rien et génère une erreur.

Dans le paramètre windows, passez une extension de fichier Windows de 1 à N caractères. Si vous ne passez pas une chaîne valide représentant une extension de fichier, la commande ne fait rien et génère une erreur.

Dans le paramètre contexte, passez la chaîne qui sera affichée dans le menu déroulant des types de fichiers dans la boîte de dialogue d'ouverture de fichiers sous Windows. La chaîne contexte est limitée à 32 caractères ; tout caractère supplémentaire est ignoré.

IMPORTANT: Une fois que vous avez associé une extension de fichier Windows à un type de fichier Mac OS, vous ne pouvez pas modifier ou supprimer cette association dans la même session de travail. Si vous avez besoin de modifier l'association pendant le développement d'une application 4D, il faut réouvrir la base et exécuter de nouveau la commande.

Exemple

La ligne de code suivante (elle peut faire partie de la Méthode base Sur ouverture) associe les fichiers de type MS-Word (sous Mac OS “WDBN”) à l'extension de fichier Windows “.DOC” :

```
ASSOCIER TYPES FICHIER ("WDBN";"DOC";"Documents Word")
```

Une fois cette ligne exécutée, le code suivant n'affiche que des documents Word dans la boîte de dialogue d'ouverture de fichiers sous Windows et Mac OS :

```
$DocRéf:=Ouvrir document("";"WDBN")
Si (OK=1)
   ` ...
Fin de si
```

Référence

Ajouter a document, Creer document, Creer fichier ressources, Ouvrir fichier ressources.

CHANGER CREATEUR DOCUMENT (document; créateur)

Paramètre	Type		Description
document	Alpha	→	Nom de document ou Chemin d'accès complet à un document
créateur	Alpha	→	Créateur de fichier (Mac OS) ou Chaîne vide (Windows)

Description

La commande CHANGER CREATEUR DOCUMENT définit le créateur du document dont vous avez passé le nom ou le chemin d'accès complet dans document.

Vous passez le nouveau créateur du document dans créateur.

Sous Windows, cette commande ne fait rien.

Reportez-vous à la description du créateur de document dans la section Présentation des documents système.

Référence

CHANGER PROPRIETES DOCUMENT, CHANGER TYPE DOCUMENT, Createur document.

CHANGER POSITION DANS DOCUMENT

version 11.3 (Modifiée)

CHANGER POSITION DANS DOCUMENT (docRef; offset{; ancre})

Paramètre	Type		Description
docRef	DocRef	→	Numéro de référence de document
offset	Réel	→	Position dans fichier (exprimée en octets)
ancre	Entier	→	1 = Par rapport au début du fichier 2 = Par rapport à la fin du fichier 3 = Par rapport à la position courante

Description

Cette commande ne fonctionne qu'avec un document déjà ouvert, dont vous avez passé le numéro de référence dans le paramètre docRef.

CHANGER POSITION DANS DOCUMENT définit la position que vous passez dans offset comme étant celle à laquelle la prochaine lecture (RECEVOIR PAQUET) ou écriture (ENVOYER PAQUET) aura lieu.

Si vous omettez le paramètre optionnel ancre, la position est définie par rapport au début du document. Sinon, vous pouvez passer dans le paramètre ancre une des valeurs listées ci-dessus.

En fonction de l'ancre définie, vous pouvez passer des valeurs positives ou négatives dans le paramètre offset.

Référence

ENVOYER PAQUET, Position dans document, RECEVOIR PAQUET.

CHANGER PROPRIETES DOCUMENT (document; verrouillé; invisible; créé le; créé à; modifié le; modifié à)

Paramètre	Type		Description
document	Alpha	→	Nom du document ou Chemin d'accès complet au document
verrouillé	Booléen	→	Verrouillé (Vrai) ou non verrouillé (Faux)
invisible	Booléen	→	Invisible (Vrai) ou visible (Faux)
créé le	Date	→	Date de création
créé à	Heure	→	Heure de création
modifié le	Date	→	Date de dernière modification
modifié à	Heure	→	Heure de dernière modification

Description

La commande CHANGER PROPRIETES DOCUMENT modifie certaines informations du document dont vous avez passé le nom ou le chemin d'accès dans document.

Avant l'appel :
• Passez Vrai dans verrouillé pour verrouiller le document. Un document verrouillé ne peut être modifié. Passez Faux dans verrouillé pour déverrouiller un document.
• Passez Vrai dans invisible pour cacher le document. Passez Faux dans invisible pour rendre le document visible dans les fenêtres du bureau.
• Passez la date et l'heure de création du document dans créé le et créé à.
• Passez la date et l'heure de la dernière modification du document dans modifié le et modifié à.

L'heure et la date de création et de dernière modification sont gérées par le gestionnaire de fichiers de votre système, à chaque fois que vous créez ou modifiez un document. Cette commande vous permet de modifier ces propriétés, dans des buts particuliers. Reportez-vous à l'exemple de la commande PROPRIETES DOCUMENT.

Référence

CHANGER CREATEUR DOCUMENT, CHANGER TYPE DOCUMENT, PROPRIETES DOCUMENT.

CHANGER TAILLE DOCUMENT

version 11.3 (Modifiée)

CHANGER TAILLE DOCUMENT (docRef; taille)

Paramètre	Type		Description
docRef	DocRef	→	Numéro de référence de document
taille	Réel	→	Nouvelle taille (en octets) de document

Description

La commande CHANGER TAILLE DOCUMENT fixe la taille d'un document au nombre d'octets que vous avez passé dans taille.

Le document doit avoir été ouvert au préalable. Vous passez son numéro de référence dans docRef.

Sous Mac OS, c'est la taille de la data fork du document qui est modifiée.

Référence

CHANGER POSITION DANS DOCUMENT, Position dans document, Taille document.

CHANGER TYPE DOCUMENT (document; type)

Paramètre	Type		Description
document	Alpha	→	Nom de document ou Chemin d'accès complet à un document
type	Alpha	→	Extension de fichier Windows ou type de fichier Mac OS (chaîne de 4 caractères)

Description

La commande CHANGER TYPE DOCUMENT définit le type du document dont vous avez passé le nom ou le chemin d'accès complet dans document.

Vous passez le nouveau type du document dans type. Reportez-vous à la description des types de documents dans les sections Présentation des documents système et Type document.

Sous Windows, la commande modifie l'extension et donc le nom du fichier décrit par document. Par exemple, l'instruction CHANGER TYPE DOCUMENT("C:\\Docs\\Facture.asc";"TEXT") renomme le fichier "Facture.asc" en "Facture.txt" (dans 4D, le type Mac "TEXT" est associé au type Windows "txt").
Si le type n'est pas associé dans 4D, il suffit de passer directement les trois lettres de l'extension. Par exemple, l'instruction CHANGER TYPE DOCUMENT("C:\\Docs\\Facture.asc";"zip") renomme le fichier "Facture.asc" en "Facture.zip".

Référence

ASSOCIER TYPES FICHIER, CHANGER CREATEUR DOCUMENT, CHANGER PROPRIETES DOCUMENT, Type document.

COPIER DOCUMENT (nomSource; nomDest{; *})

Paramètre	Type		Description
nomSource	Alpha	→	Nom du document à copier
nomDest	Alpha	→	Nom du document copié
*		→	Remplacer document existant le cas échéant

Description

La commande COPIER DOCUMENT copie le document désigné par nomSource à l'emplacement désigné par nomDest.

Les deux paramètres nomSource et nomDest peuvent désigner un document situé dans le dossier de la base, ou un chemin d'accès complet exprimé par rapport à la racine du volume.

Une erreur est générée si un document nommé nomDest existe déjà, à moins que vous n'ayez spécifié le paramètre optionnel *, qui indique à COPIER DOCUMENT de supprimer et de remplacer le document existant par le document de destination dans ce cas.

Exemples

(1) L'exemple suivant duplique un document dans son propre dossier :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"C:\\DOSSIER\\LeDoc2")
```

(2) L'exemple suivant copie un document dans le dossier de la base (dans la mesure où C:\\DOSSIER n'est pas le dossier de la base) :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"LeDoc")
```

(3) L'exemple suivant copie un document d'un volume vers un autre :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"F:\\Archives\\LeDoc.OLD")
```

(4) L'exemple suivant duplique un document dans son propre dossier, écrasant la précédente copie si elle existe :

```
COPIER DOCUMENT("C:\\DOSSIER\\LeDoc";"C:\\DOSSIER\\LeDoc2";*)
```

Référence

DEPLACER DOCUMENT.

Createur document

Createur document (document) → Alpha

Paramètre	Type		Description
document	Alpha	→	Nom de document ou Chemin d'accès complet à un document
Résultat	Alpha	←	Chaîne vide (Windows) ou Créateur de fichier (Mac OS)

Description

La commande Createur document retourne le "créateur" du document dont vous avez passé le nom ou le chemin d'accès complet dans document.

Sous Windows, Createur document retourne une chaîne vide.

Référence

CHANGER CREATEUR DOCUMENT, Type document.

CREER ALIAS (cheminCible; cheminAlias)

Paramètre	Type		Description
cheminCible	Alpha	→	Nom ou chemin d'accès de la cible de l'alias/du raccourci
cheminAlias	Alpha	→	Nom ou chemin d'accès complet de l'alias/du raccourci à créer

Description

La commande CREER ALIAS crée un alias (appelé "raccourci" sous Windows) du fichier ou dossier cible désigné par le paramètre cheminCible, avec le nom et l'emplacement définis dans le paramètre cheminAlias.

Vous pouvez créer un alias de tout type de document ou de dossier. L'icône de l'alias sera identique à celle de l'élément cible. Elle comportera en outre une petite flèche et, sous Mac OS, le libellé de l'alias apparaîtra en caractères italiques.

La commande n'affecte pas de libellé par défaut à l'alias, vous devez passer un nom dans le paramètre cheminAlias. Si vous passez uniquement un nom dans ce paramètre, l'alias est créé dans le dossier actif courant (généralement, le dossier contenant le fichier de structure de la base).

Note : Sous Windows, les raccourcis sont des fichiers dont l'extension est ".LNK". Si vous ne passez pas cette extension, la commande l'ajoute automatiquement.

Si vous passez une chaîne vide dans cheminCible, la commande ne fait rien.

Exemple

Votre base génère des fichiers texte intitulés "Rapport*Numéro*", stockés dans le dossier de la base. Vous souhaitez permettre à l'utilisateur de créer des raccourcis vers ces rapports et de les stocker où il le souhaite :

```
      `Méthode CREER_RAPPORT
C_TEXTE($vtRapport)
C_ALPHA(250;$vtChemin)
C_ALPHA(80;$vaNom)
C_HEURE(vDoc)
C_ENTIER($NumRapport)
```

```
$vTRapport:=...  `Edition du rapport
$NumRapport:=...  `Calcul du numéro du rapport
$vaNom:="Rapport"+Chaine($NumRapport)+".txt"  `Nom du fichier
vDoc:=Creer document($vaNom)
Si (OK=1)
    ENVOYER PAQUET(vDoc;$vTRapport)
    FERMER DOCUMENT(vDoc)
        `Ajout de l'alias
    CONFIRMER("Créer un alias pour ce rapport ?)
    Si (OK=1)
        $vtChemin:=Selectionner dossier ("Où souhaitez-vous créer l'alias ?")
        Si (OK=1)
            CREER ALIAS ($vaNom;$vtChemin+$vaNom)
            Si (OK=1)
                MONTRER SUR DISQUE($vtChemin+$vaNom)
                    `Visualisation de l'emplacement de l'alias
            Fin de si
        Fin de si
    Fin de si
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été correctement exécutée, sinon elle prend la valeur 0.

Référence

RESOUDRE ALIAS.

Creer document (document{; leType}) → DocRef

Paramètre	Type		Description
document	Alpha	→	Nom de document ou Chemin d'accès complet de document ou Chaîne vide pour afficher la boîte de dialogue standard d'enregistrement de fichiers
leType	Alpha	→	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
Résultat	DocRef	←	Numéro de référence du document

Description

La commande Creer document crée un document et retourne son numéro de référence de document.

Vous passez le nom ou le chemin d'accès complet du nouveau document dans document. Si document existe déjà, il est remplacé. Cependant, si le document est verrouillé ou est déjà ouvert, une erreur est générée.

Si vous passez une chaîne vide dans document, une boîte de dialogue standard d'enregistrement de fichiers apparaît et l'utilisateur peut spécifier le nom du document. Si dans ce cas l'utilisateur clique sur le bouton **Annuler**, Creer document retourne une référence de document nulle, et la variable OK prend la valeur 0.

Creer document crée par défaut un document de type TEXT (Mac OS) ou .TXT (Windows). Pour créer un autre type de document, passez un type dans le paramètre optionnel leType.

Si vous utilisez la boîte de dialogue standard d'enregistrement de fichiers, vous pouvez passer dans le paramètre leType un ou plusieurs type(s) de fichier(s) afin de configurer la liste des types autorisés dans la boîte de dialogue. Vous pouvez passer une liste de plusieurs types séparés par un ; (point virgule). Pour chaque type défini, une ligne sera ajoutée dans le menu de choix de type de la boîte de dialogue.

Sous Mac OS, vous pouvez passer soit type Mac OS classique (TEXT, APPL, etc.), soit un type UTI (Uniform Type Identifier). Les types UTIs ont été définis par Apple afin de répondre aux besoins d'uniformisation des types de fichiers. Par exemple, "public.text" est le type UTI des fichiers de type texte. Pour plus d'informations sur les UTIs, reportez-vous à l'adresse http://developer.apple.com/documentation/Carbon/Conceptual/understanding_utis/index.html (documentation en anglais).
Sous Windows, vous pouvez également passer un type de fichier classique Mac OS — 4D effectue la correspondance en interne — ou l'extension des fichiers (.txt, .exe, etc.). A noter que sous Windows, l'utilisateur aura la possibilité de "forcer" l'affichage de tous les types de fichiers en saisissant ***.*** dans la boîte de dialogue. Toutefois dans ce cas, 4D effectuera une vérification supplémentaire des types des fichiers sélectionnés : si l'utilisateur sélectionne un type de fichier non autorisé, la commande retourne une erreur.
Si vous ne souhaitez pas restreindre les fichiers affichés à un ou plusieurs types, passez la chaîne "*" (étoile) ou ".*" dans leType.
Sous Windows, vous pouvez passer une extension de fichier Windows ou un type de fichier Mac OS associé à l'aide de la commande ASSOCIER TYPES FICHIER. Si vous souhaitez créer un document sans extension, un document comportant plusieurs extensions, ou un document comportant une extension de plus de trois caractères, n'utilisez pas le paramètre leType et passez le nom complet dans document (cf. exemple 2).

Si le document est correctement créé et ouvert, Creer document retourne sa référence de document et la variable système OK prend la valeur 1. La variable système Document est mise à jour et retourne le chemin d'accès complet du document créé.

Une fois que vous avez créé et ouvert un document, vous pouvez écrire ou lire des valeurs dans ce document à l'aide des commandes RECEVOIR PAQUET et ENVOYER PAQUET, que vous pouvez combiner avec les commandes Position dans document et CHANGER POSITION DANS DOCUMENT pour accéder directement à certains endroits du document.

N'oubliez pas d'appeler finalement FERMER DOCUMENT pour le document.

Exemples

(1) L'exemple suivant crée et ouvre un nouveau document qui s'appelle "Note", écrit la chaîne "Bonjour" et le referme :

```
C_HEURE(vDoc)
vDoc := Creer document ("Note.txt")  ` Créer un nouveau document qui s'appelle Note
Si (OK=1)
    ENVOYER PAQUET (vDoc; "Bonjour")  ` Ecrire un mot dans le document
    FERMER DOCUMENT (vDoc)  ` Fermer le document
Fin de si
```

(2) L'exemple suivant crée sous Windows des documents avec des extensions non standard :

```
$vhMonDoc:=Creer document("LeDoc.ext1.ext2") `Plusieurs extensions
$vhMonDoc:=Creer document("LeDoc.shtml") `Extension longue
$vhMonDoc:=Creer document("LeDoc.") `Pas d'extension (le point "." est obligatoire)
```

Variables et ensembles système

Si le document est correctement créé, la variable système OK prend la valeur 1 et la variable système Document contient le chemin d'accès et le nom du fichier document.

Référence

Ajouter a document, Ouvrir document.

CREER DOSSIER (cheminAccès)

Paramètre	Type		Description
cheminAccès	Alpha	→	Chemin d'accès au nouveau dossier à créer

Description

La commande CREER DOSSIER crée un dossier en fonction du chemin d'accès que vous passez dans le paramètre cheminAccès.

Si vous passez un nom, le dossier est créé dans le dossier de la base. Si vous passez un chemin d'accès complet, cela doit être un chemin d'accès valide au nom du dossier à créer, à partir de la racine du volume ou du dossier de la base.

Exemples

(1) L'exemple suivant crée le dossier “Archives” dans le dossier de la base :

```
CREER DOSSIER("Archives")
```

(2) L'exemple suivant crée le dossier “Archives” dans le dossier de la base, puis crée les sous-dossiers “Janvier” et “Février”:

```
CREER DOSSIER("Archives")
CREER DOSSIER("Archives\\Janvier")
CREER DOSSIER("Archives\\Février")
```

(3) L'exemple suivant crée le dossier “Archives” à la racine du volume C :

```
CREER DOSSIER("C:\\Archives")
```

(4) L'exemple suivant échouera s'il n'existe pas déjà de dossier “Nouveautés” à la racine du volume C :

```
CREER DOSSIER("C:\\Nouveautés\\Images")
   ` INCORRECT, on ne peut pas créer deux niveaux de dossier en un seul appel
```

Référence

LISTE DES DOSSIERS, Tester chemin acces.

DEPLACER DOCUMENT (cheminSource; cheminDest)

Paramètre	Type		Description
cheminSource	Alpha	→	Chemin d'accès complet au document existant
cheminDest	Alpha	→	Chemin d'accès de destination

Description

La commande DEPLACER DOCUMENT déplace ou renomme un document.

Vous passez le chemin d'accès complet au document existant dans le paramètre cheminSource et le nouveau nom et/ou emplacement du document dans cheminDest.

Attention : Avec DEPLACER DOCUMENT, vous pouvez déplacer un document depuis et vers tous les dossiers du même volume. Si vous souhaitez déplacer un document entre deux volumes différents, utilisez la commande COPIER DOCUMENT pour “déplacer” le document puis effacez le document original avec la commande SUPPRIMER DOCUMENT.

Exemples

(1) L'exemple suivant renomme le document DocNom :

```
DEPLACER DOCUMENT("C:\\DOSSIER\\DocNom";"C:\\DOSSIER\\NouveauDocNom")
```

(2) L'exemple suivant déplace et renomme le document DocNom :

```
DEPLACER DOCUMENT("C:\\DOSSIER1\\DocNom";"C:\\DOSSIER2\\
                                                NouveauDocNom")
```

(3) L'exemple suivant déplace le document DocNom :

```
DEPLACER DOCUMENT("C:\\DOSSIER1\\DocNom";"C:\\DOSSIER2\\DocNom")
```

Note : Dans les deux derniers exemples, le dossier de destination "C:\\DOSSIER2" doit déjà exister. En effet, la commande DEPLACER DOCUMENT déplace uniquement un document, elle ne peut créer de dossiers.

Référence

COPIER DOCUMENT.

FERMER DOCUMENT (réfDocument)

Paramètre	Type		Description
docRef	DocRef	→	Numéro de référence du document

Description

FERMER DOCUMENT ferme le document spécifié par réfDocument.

Fermer un document est le seul moyen de s'assurer que les données écrites dans le fichier sont sauvegardées. Vous devez fermer tous les documents ouverts par les commandes Ouvrir document, Creer document et Ajouter a document.

Exemple

L'exemple suivant permet à l'utilisateur de créer un nouveau document, écrit la chaîne “Bonjour”, puis le referme :

```
C_HEURE(vDoc)
vDoc := Creer document ("")
Si (OK=1)
    ENVOYER PAQUET (vDoc; "Bonjour")  ` Ecrire un mot dans le document
    FERMER DOCUMENT (vDoc)  ` Fermer le document
Fin de si
```

Référence

Ajouter a document, Creer document, Ouvrir document.

LIRE ICONE DOCUMENT (cheminDoc; icône{; taille})

Paramètre	Type		Description
cheminDoc	Alpha	→	Nom ou chemin d'accès du fichier duquel obtenir l'icône ou chaîne vide pour afficher la boîte de dialogue d'ouverture de fichiers
icône	Image	→	Champ ou variable image
		←	Icône du document
taille	Entier long	→	Taille de l'icône (en pixels)

Description

La commande LIRE ICONE DOCUMENT retourne dans le champ ou la variable image 4D icône, l'icône du document dont vous avez passé le nom ou le chemin d'accès complet dans cheminDoc. cheminDoc peut désigner un fichier de tout type (document, exécutable, raccourci ou alias...) ou un dossier.

Passez dans cheminDoc le chemin d'accès absolu du document dont vous souhaitez récupérer l'icône. Vous pouvez passer uniquement le nom du document ou un chemin d'accès relatif, dans ce cas il doit se trouver dans le dossier courant de la base (généralement, le dossier contenant le fichier de structure de la base).
Si vous passez une chaîne vide dans cheminDoc, la boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de désigner un fichier. Une fois la boîte de dialogue validée, la variable système *Document* contient le chemin d'accès complet du fichier sélectionné.

Passez dans le paramètre icône un champ ou une variable image 4D. Après l'exécution de la commande, ce paramètre contient l'icône du fichier (au format PICT).

Le paramètre optionnel taille vous permet d'indiquer les dimensions de l'image que vous souhaitez obtenir. La valeur du paramètre correspond à la longueur d'un côté du carré dans lequel l'image sera incluse. Généralement, les icônes sont définies en 32x32 pixels ("grande icône") ou 16x16 pixels ("petite icône"). Si vous passez 0 ou omettez le paramètre, la commande retourne l'icône dans sa plus grande taille disponible.

LISTE DES DOCUMENTS (cheminAccès; documents)

Paramètre	Type		Description
cheminAccès	Chaîne	→	Chemin d'accès de volume ou de dossier
documents	Tab chaîne	←	Nom des documents situés à cet endroit

Description

La commande LISTE DES DOCUMENTS remplit le tableau de type Texte ou Alpha documents avec les noms des documents situés à l'endroit que vous avez indiqué avec le paramètre cheminAccès.

Note : Vous devez passer un chemin d'accès absolu dans le paramètre cheminAccès.

S'il n'y pas de document à cet endroit, la commande retourne un tableau vide. Si le chemin d'accès que vous avez passé dans cheminAccès est invalide, LISTE DES DOCUMENTS génère une erreur de gestionnaire de fichier que vous pouvez intercepter à l'aide d'une méthode installée par APPELER SUR ERREUR.

Référence

LISTE DES DOSSIERS, LISTE DES VOLUMES.

LISTE DES DOSSIERS (cheminAccès; dossiers)

Paramètre	Type		Description
cheminAccès	Chaîne	→	Chemin d'accès de volume, répertoire ou dossier
dossiers	Tableau chaîne	←	Noms des dossiers situés à cet endroit

Description

La commande LISTE DES DOSSIERS remplit le tableau de type Texte ou Alpha dossiers avec les noms des dossiers (répertoires sous Windows) situés à l'endroit que vous avez indiqué avec le paramètre cheminAccès.

Note : Vous devez passer un chemin d'accès absolu dans le paramètre cheminAccès.

S'il n'y pas de dossier à cet endroit, la commande retourne un tableau vide. Si le chemin d'accès que vous avez passé dans cheminAccès est invalide, LISTE DES DOSSIERS génère une erreur de gestionnaire de fichiers que vous pouvez intercepter à l'aide d'une méthode installée par APPELER SUR ERREUR.

Référence

LISTE DES DOCUMENTS, LISTE DES VOLUMES.

LISTE DES VOLUMES (volumes)

Paramètre	Type		Description
volumes	Tableau chaîne	←	Noms des volumes actuellement montés

Description

LISTE DES VOLUMES remplit le tableau volumes, de type Texte ou Alpha, avec les noms des volumes définis (Windows) ou montés (Mac OS) sur votre machine.

Sous Mac OS, elle retourne la liste des volumes visibles au niveau du Finder.

En revanche, sous Windows, elle retourne la liste des volumes couramment définis, même si le volume n'est pas physiquement présent (par exemple le volume "A:\" sera retourné même s'il n'y a pas de disquette dans le lecteur).

Exemple

A l'aide de la zone de défilement taVolumes, vous voulez afficher la liste des volumes définis ou montés sur votre machine :

```
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU ALPHA(31;taVolumes;0)
        LISTE DES VOLUMES(taVolumes)

        ` ...
Fin de cas
```

Référence

LISTE DES DOCUMENTS, LISTE DES DOSSIERS, PROPRIETES DU VOLUME.

MONTRER SUR DISQUE (cheminAccès{; *})

Paramètre	Type		Description
cheminAccès	Chaîne	→	Chemin d'accès de l'élément à montrer
*		→	Si l'élément est un dossier, montrer son contenu

Description

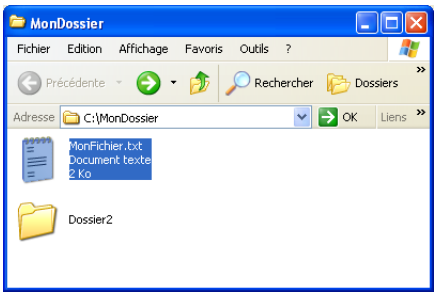
La commande MONTRER SUR DISQUE affiche dans une fenêtre standard du système d'exploitation le fichier ou le dossier dont le chemin d'accès est passé dans le paramètre cheminAccès.
Dans le cadre d'une interface utilisateur, cette commande permet à l'utilisateur de visualiser l'emplacement d'un fichier ou d'un dossier spécifique.

Par défaut, si cheminAccès désigne un dossier, la commande affiche le niveau du dossier lui-même. Si vous passez le paramètre facultatif *, la commande ouvre le dossier et affiche son contenu dans la fenêtre. Si cheminAccès désigne un fichier, le paramètre * est ignoré.

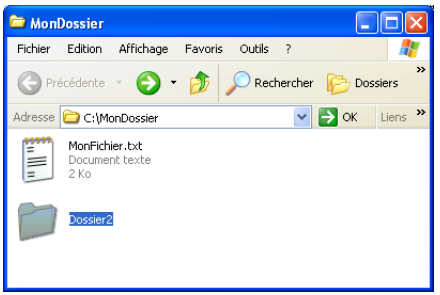
Exemples

Ces exemples illustrent le fonctionnement de la commande.

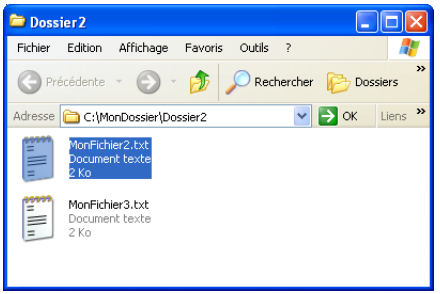
```
MONTRER SUR DISQUE("c:\\MonDossier\\MonFichier.txt")  `Affiche le fichier désigné
```



MONTRER SUR DISQUE("c:\\MonDossier\\Dossier2") `Affiche le dossier désigné



MONTRER SUR DISQUE("c:\\MonDossier\\Dossier2";*) `Affiche le contenu du dossier désigné



Variables et ensembles système

La variable système OK prend la valeur 1 si la commande est correctement exécutée, sinon elle prend la valeur 0.

Ouvrir document (document{; leType{; mode}}) → DocRef

Paramètre	Type		Description
document	Alpha	→	Nom du document ou Chemin d'accès complet au document ou Chaîne vide pour afficher la boîte de dialogue
leType	Alpha	→	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
mode	Entier	→	Mode d'ouverture du document
Résultat	DocRef	←	Numéro de référence du document

Description

La commande Ouvrir document ouvre le document dont vous avez passé le nom dans document.

Si vous passez une chaîne vide ("") dans document, une boîte de dialogue standard d'ouverture de fichiers apparaît et l'utilisateur peut désigner le document. Si dans ce cas l'utilisateur clique sur le bouton **Annuler**, aucun document n'est ouvert, Ouvrir document retourne une référence de document nulle, et la variable OK prend la valeur 0.

• Si le document est correctement ouvert, Ouvrir document retourne sa référence de document et la variable OK prend la valeur 1.
• Si le document était déjà ouvert et que le paramètre mode n'est pas précisé, Ouvrir document l'ouvre en mode lecture et la variable OK prend la valeur 1.
• Si le document était déjà ouvert et que vous tentez de l'ouvrir en mode écriture, une erreur est générée.
• Si le document n'existe pas, une erreur est générée.

Passez dans le paramètre leType le ou les type(s) de fichier(s) pouvant être sélectionnés dans la boîte de dialogue d'ouverture. Vous pouvez passer une liste de plusieurs types séparés par un ; (point virgule). Pour chaque type défini, une ligne sera ajoutée dans le menu de choix de type de la boîte de dialogue.

Sous Mac OS, vous pouvez passer soit type Mac OS classique (TEXT, APPL, etc.), soit un type UTI (Uniform Type Identifier). Les types UTIs ont été définis par Apple afin de répondre aux besoins d'uniformisation des types de fichiers. Par exemple, "public.text" est le type UTI des fichiers de type texte. Pour plus d'informations sur les UTIs, reportez-vous à l'adresse http://developer.apple.com/documentation/Carbon/Conceptual/understanding_utis/index.html (documentation en anglais).
Sous Windows, vous pouvez également passer un type de fichier classique Mac OS — 4D effectue la correspondance en interne — ou l'extension des fichiers (.txt, .exe, etc.). A noter que sous Windows, l'utilisateur aura la possibilité de "forcer" l'affichage de tous les types de fichiers en saisissant ***.*** dans la boîte de dialogue. Toutefois dans ce cas, 4D effectuera une vérification supplémentaire des types des fichiers sélectionnés : si l'utilisateur sélectionne un type de fichier non autorisé, la commande retourne une erreur.
Si vous ne souhaitez pas restreindre les fichiers affichés à un ou plusieurs types, passez la chaîne "*" (étoile) ou ".*" dans leType.

Le paramètre optionnel mode permet de définir le mode d'ouverture du fichier document. Quatre modes d'ouverture sont disponibles. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème Documents système :

Constante	Type	Valeur
Lecture et écriture (valeur par défaut)	Entier	0
Mode écriture	Entier	1
Mode lecture	Entier	2
Lire chemin accès	Entier	3

Lorsqu'un document est ouvert, Ouvrir document se place initialement au début du document, alors que Ajouter a document se place à la fin.

Une fois que vous avez ouvert un document, vous pouvez écrire ou lire des valeurs dans ce document à l'aide des commandes RECEVOIR PAQUET et ENVOYER PAQUET, que vous pouvez combiner avec les commandes Position dans document et CHANGER POSITION DANS DOCUMENT pour accéder directement à certains endroits du document.

N'oubliez pas d'appeler finalement FERMER DOCUMENT pour le document.

Exemples

(1) L'exemple suivant ouvre un document existant qui s'appelle “Note”, écrit la chaîne “Au revoir” dans le document et le referme. Si le document contient déjà la chaîne “Bonjour”, elle est remplacée :

```
C_HEURE(vDoc)
vDoc := Ouvrir document ("Note.txt";Lecture et écriture) ` Ouvrir le document Note
Si (OK=1)
    ENVOYER PAQUET (vDoc; "Au revoir") ` Ecrire un mot dans le document
    FERMER DOCUMENT (vDoc) ` Fermer le document
Fin de si
```

(2) Vous pouvez lire un document déjà ouvert en écriture :

```
vDoc:=Ouvrir document ("PassFile";"TEXT")     ` Le fichier est ouvert
vRef:=Ouvrir document ("PassFile";"TEXT";Mode lecture) ` Le fichier est lu
```

Variables et ensembles système

Si le document est correctement ouvert, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Après l'appel, la variable système Document contient le nom complet du document.
Si vous passez la valeur 3 dans mode, la fonction retourne ?00:00:00? (pas de référence de document). Le document n’est pas ouvert mais les variables système Document et OK sont mises à jour :
• OK prend la valeur 1,
• Document contient le chemin d’accès et le nom du fichier document.

Note : Si vous passez une chaîne vide dans document, une boîte de dialogue d’ouverture de fichiers apparaît. Si elle est validée, Document et OK sont mises à jour comme décrit ci-dessus. Si elle est annulée, OK prend la valeur 0.

Référence

Ajouter a document, Creer document.

Position dans document

Position dans document (docRef) → Numérique

Paramètre	Type		Description
docRef	DocRef	→	Numéro de référence de document
Résultat	Numérique	←	Position dans le fichier (exprimée en octets) à partir du début du fichier

Description

Cette commande ne fonctionne qu'avec un document déjà ouvert, dont vous avez passé le numéro de référence dans le paramètre docRef.

Position dans document retourne la position, à partir du début du document, à laquelle la prochaine lecture (RECEVOIR PAQUET) ou écriture (ENVOYER PAQUET) aura lieu.

Référence

CHANGER POSITION DANS DOCUMENT, ENVOYER PAQUET, RECEVOIR PAQUET.

PROPRIETES DOCUMENT (document; verrouillé; invisible; créé le; créé à ; modifié le; modifié à)

Paramètre	Type		Description
document	Alpha	→	Nom du document
verrouillé	Booléen	←	Verrouillé (Vrai) ou non verrouillé (Faux)
invisible	Booléen	←	Invisible (Vrai) ou visible (Faux)
créé le	Date	←	Date de création
créé à	Heure	←	Heure de création
modifié le	Date	←	Date de la dernière modification
modifié à	Heure	←	Heure de la dernière modification

Description

La commande PROPRIETES DOCUMENT retourne des informations sur le document dont le nom ou le chemin d'accès est passé dans le paramètre document.

Après l'appel :
• verrouillé retourne Vrai si le document est verrouillé. Un document verrouillé ne peut pas être modifié.
• invisible retourne Vrai si le document est caché.
• créé le et créé à retournent la date et l'heure de création du document.
• modifié le et modifié à retournent la date et l'heure de la dernière modification du document.

Exemples

Vous avez créé une base de documentation et vous voulez exporter tous les enregistrements créés dans le base vers un document sur disque. Comme la base est régulièrement mise à jour, vous voulez écrire un algorithme d'export qui crée ou recrée chaque document sur disque si le document n'existe pas ou si l'enregistrement correspondant a été modifié depuis la dernière sauvegarde du document. Par conséquent, vous devez comparer la date et l'heure de modification du document (s'il existe) avec celles de l'enregistrement correspondant.

Pour illustrer cet exemple, nous allons utiliser la table suivante :

Documents	
Numéro	**L**
Sujet	**A**
Thème	**A**
Description	**T**
Marqueur création	**L**
Marqueur modification	**L**

Plutôt que de sauvegarder une date et une heure dans chaque enregistrement, vous pouvez stocker un "marqueur" dont la valeur exprime le nombre de secondes écoulées depuis une date antérieure arbitraire (dans cet exemple, le 1er janvier 1995 à 00:00:00) ainsi que la date et l'heure de la sauvegarde de l'enregistrement.

Dans notre exemple, le champ [Documents]Marqueur création contient le marqueur de création de l'enregistrement et le champ [Documents]Marqueur modification contient le marqueur de la dernière modification de l'enregistrement.

La méthode projet marqueurTemps suivante calcule le marqueur de temps par rapport à une date et une heure spécifiques ou par rapport à la date et l'heure courantes si aucun paramètre n'est passé :

```
` Méthode projet marqueurTemps
` marqueurTemps { ( Date ; Heure ) } -> Entier long
` marqueurTemps { ( Date ; Heure ) } -> Nombre de secondes depuis le 1er janvier 1995

C_DATE($1;$vdDate)
C_HEURE($2;$vhTime)
C_ENTIER LONG($0)

Si (Nombre de parametres=0)
   $vdDate:=Date du jour
   $vhTime:=Heure courante
Sinon
   $vdDate:=$1
   $vhTime:=$2
Fin de si
$0:=(($vdDate-!01/01/95!)*86400)+$vhTime
```

Note : Avec cette méthode, vous pouvez encoder toutes les dates et les heures situées entre le 01/01/95 à 00:00:00 et le 19/01/2063 à 03:14:07, ce qui représente l'intervalle de données exploitables par un entier long (de 0 à 2^31 moins 1).

A l'inverse, les méthodes projet Marqueur vers date et Marqueur vers heure vous permettent d'extraire la date et l'heure stockées dans un marqueur :

```
    ` Méthode projet Marqueur vers date
    ` Marqueur vers date ( Entier long ) -> Date
    ` Marqueur vers date ( Marqueur ) -> Date extraite

C_DATE($0)
C_ENTIER LONG($1)

$0:=!01/01/95!+($1\86400)
```

```
    ` Méthode projet Marqueur vers heure
    ` Marqueur vers heure ( Entier long ) -> Heure
    ` Marqueur vers heure ( Marqueur ) -> Heure extraite

C_HEURE($0)
C_ENTIER LONG($1)

$0:=Heure(Chaine heure(†00:00:00†+($1%86400)))
```

Pour vous assurer que les marqueurs des enregistrements sont correctement mis à jour, quelle que soit la manière dont ils sont créés ou modifiés, il suffit de faire appliquer cette règle par le trigger de la table [Documents]:

```
    ` Trigger de la table [Documents]

Au cas ou
    : (Evenement moteur=Sauvegarde nouvel enreg)
        [Documents]Marqueur création:=marqueurTemps
        [Documents]Marqueur modification:=marqueurTemps
    : (Evenement moteur=Sauvegarde enregistrement)
        [Documents]Marqueur modification:=marqueurTemps
Fin de cas
```

Une fois que cela est implémenté dans votre base, il suffit d'écrire la méthode projet CREER DOCUMENTATION listée ci-dessous. Nous utilisons PROPRIETES DOCUMENT et CHANGER PROPRIETES DOCUMENT pour gérer la date et l'heure de création et de modification des documents.

```
    ` Méthode projet CREER DOCUMENTATION

C_ALPHA(255;$vsPath;$vsDocPathName;$vsDocName)
C_ENTIER LONG($vlDoc)
C_BOOLEEN($vbOnWindows;$vbDoIt;$vbLocked;$vbInvisible)
C_HEURE($vhDocRef;$vhCreatedAt;$vhModifiedAt)
C_DATE($vdCreatedOn;$vdModifiedOn)

Si (Type application=4D Client)
        ` Si 4D Client est utilisé, sauvegarder les documents localement
        ` c'est-à-dire sur le poste client où se trouve 4D Client
    $vsPath:=Nom long vers chemin d'accès (Type application)
Sinon
        ` Sinon, sauvegarder les documents là où se trouve le fichier de données
    $vsPath:=Nom long vers chemin d'accès (Fichier donnees)
Fin de si
    ` Stocker les documents dans un répertoire nommé arbitrairement "Documentation"
$vsPath:=$vsPath+"Documentation"+Caractere(Symbole séparateur)
    ` Si ce répertoire n'existe pas, le créer
Si (Tester chemin acces ($vsPath) # Est un répertoire)
    CREER DOSSIER($vsPath)
Fin de si
    ` Etablir la liste des documents existants
    ` car nous allons devoir supprimer ceux qui sont obsolètes, autrement dit
    ` ceux dont les enregistrements correspondants ont été supprimés.
TABLEAU ALPHA(255;$asDocument;0)
LISTE DES DOCUMENTS($vsPath;$asDocument)
    ` Sélection de tous les enregistrements de la table [Documents]
TOUT SELECTIONNER([Documents])
    ` Pour chaque enregistremnt
$vlNbRecords:=Enregistrements trouves([Documents])
$vlNbDocs:=0
$vbOnWindows:=Sous Windows
```

```
Boucle ($vlDoc;1;$vlNbRecords)
        ` Supposons que nous aurons à (re)créer le document sur disque
    $vbDoIt:=Vrai
        ` Calcul du nom et du chemin d'accès au document
    $vsDocName:="DOC"+Chaine([Documents]Numéro;"00000")
    $vsDocPathName:=$vsPath+$vsDocName
        ` Est-ce que ce document existe déjà ?
    Si (Tester chemin acces($vsDocPathName+".HTM")=Est un document)
            ` Si oui, retirer le document de la liste des documents
            ` qui peuvent être supprimés
        $vlElem:=Chercher dans tableau($asDocument;$vsDocName+".HTM")
        Si ($vlElem>0)
            SUPPRIMER DANS TABLEAU($asDocument;$vlElem)
        Fin de si
            ` Est-ce que le document a été stocké après la dernière modification
            `de l'enregistrement?
        PROPRIETES DOCUMENT($vsDocPathName+".HTM";$vbLocked;$vbInvisible;
                                $vdCreatedOn;$vhCreatedAt;$vdModifiedOn;$vhModifiedAt)
        Si (marqueurTemps ($vdModifiedOn;$vhModifiedAt)>=
                                                    [Documents]Marqueur modification)
                ` Si oui, nous n'avons pas besoin de recréer le document
            $vbDoIt:=Faux
        Fin de si
    Sinon
            ` Le document n'existe pas, mettre ces deux variables à zéro, pour que
            ` nous sachions que nous devrons les traiter avant de fixer les propriétés finales
            ` du document
        $vdModifiedOn:=!00/00/00!
        $vhModifiedAt:=†00:00:00†
    Fin de si
        ` Avons-nous besoin de (re)créer le document?
    Si ($vbDoIt)
            ` Si oui, incrémenter le nombre de documents mis à jour
        $vlNbDocs:=$vlNbDocs+1
            ` Supprimer le document s'il existe déjà
        SUPPRIMER DOCUMENT($vsDocPathName+".HTM")
            ` Et le recréer
        Si ($vbOnWindows)
            $vhDocRef:=Creer document($vsDocPathName;"HTM")
        Sinon
```

```
            $vhDocRef:=Creer document($vsDocPathName+".HTM")
        Fin de si
        Si (OK=1)
            ` ...
            ` Ecrivons ici le contenu du document
            ` ...
            FERMER DOCUMENT($vhDocRef)
            Si ($vdModifiedOn=!00/00/00!)
                ` Le document n'existait pas, fixer les valeurs correctes pour
                ` la date et l'heure de modification
                $vdModifiedOn:=Date du jour
                $vhModifiedAt:=Heure courante
            Fin de si
            ` Changer les propriétés du document de telle manière que sa date et son
            `heure de création soit égales à celles de l'enregistrement correspondant
            CHANGER PROPRIETES DOCUMENT($vsDocPathName+".HTM";$vbLocked;
                    $vbInvisible;Marqueur vers date ([Documents]Marqueur création);
                                 Marqueur vers heure ([Documents]Marqueur création);
                                 $vdModifiedOn;$vhModifiedAt)
        Fin de si
    Fin de si
    ` Juste pour savoir ce qui se passe
    CHANGER TITRE FENETRE("Traitement du document "+Chaine($vlDoc)+" sur "
                          +Chaine($vlNbRecords))
    ENREGISTREMENT SUIVANT([Documents])
Fin de boucle
` Suppression des documents obsolètes, c'est-à-dire ceux
` qui sont toujours dans le tableau $asDocument
Boucle ($vlDoc;1;Taille tableau($asDocument))
    SUPPRIMER DOCUMENT($vsPath+$asDocument{$vlDoc})
    CHANGER TITRE FENETRE("Suppression du document obsolète: "+Caractere(34)+
                          $asDocument{$vlDoc}+Caractere(34))
Fin de boucle
` C'est la fin
ALERTE("Nombre de documents traités : "+Chaine($vlNbRecords)+Caractere(13)+
       "Nombre de documents mis à jour : "+Chaine($vlNbDocs)+Caractere(13)+
       "Nombre de documents supprimés : "+Chaine(Taille tableau($asDocument)))
```

Référence

CHANGER PROPRIETES DOCUMENT, Createur document, Type document.

PROPRIETES DU VOLUME (volume; taille; utilisé; libre)

Paramètre	Type		Description
volume	Alpha	→	Nom du volume
taille	Numérique	←	Taille du volume exprimée en octets
utilisé	Numérique	←	Place utilisée sur le volume exprimée en octets
libre	Numérique	←	Place libre sur le volume exprimée en octets

Description

La commande PROPRIETES DU VOLUME retourne la taille, la place utilisée et la place libre sur le volume dont le nom est passé dans volume. Ces valeurs sont exprimées en octets.

Note : Si volume indique un volume distant non monté, la variable OK prend la valeur 0 et les trois paramètres retournent -1.

Exemple

Votre application comprend des opérations par lots qui sont exécutées la nuit ou pendant le week-end. Ces opérations stockent des fichiers temporaires sur disque. Pour que cette méthode soit aussi autonome et souple que possible, vous écrivez une routine qui va automatiquement chercher et utiliser le premier volume ayant de la place disponible pour les fichiers temporaires. Voici la méthode :

```
` Méthode projet Chercher volume pour place
` Chercher volume pour place ( Reel ) -> Alpha
` Chercher volume pour place ( Place nécessaire en octets ) ->
`Nom du volume ou chaîne vide

C_ALPHA(31;$0)
C_ALPHA(255;$vaNomDoc)
C_ENTIER LONG($vlNbVolumes;$vlVolume)
C_REEL($1;$vlTaille;$vlUtilisé;$vlLibre)
C_HEURE($vhDocRef)
```

```
    ` Initialiser le résultat de la fonction
$0:=""
    ` Protéger toutes les opérations d'entrée/sortie par une méthode d'interruption d'erreur
APPELER SUR ERREUR("METHODE ERREUR")
    ` Obtenir la liste des volumes
TABLEAU ALPHA(31;$taVolumes;0)
gErreur:=0
LISTE DES VOLUMES($taVolumes)
Si (gErreur=0)
        ` Si nous sommes sous Windows, ignorer les deux lecteurs de disquettes
    Si (Sous Windows )
        $vlVolume:=Chercher dans tableau($taVolumes;"A:\")
        Si ($vlVolume>0)
            SUPPRIMER DANS TABLEAU($taVolumes;$vlVolume)
        Fin de si
        $vlVolume:=Chercher dans tableau($taVolumes;"B:\")
        Si ($vlVolume>0)
            SUPPRIMER DANS TABLEAU($taVolumes;$vlVolume)
        Fin de si
    Fin de si
    $vlNbVolumes:=Taille tableau($taVolumes)
        ` Pour chaque volume
    Boucle ($vlVolume;1;$vlNbVolumes)
            ` Obtenir la taille, la place utilisée et la place libre
        gErreur:=0
        PROPRIETES DU VOLUME($taVolumes{$vlVolume};$vlTaille;$vlUtilisé;$vlLibre)
        Si (gErreur=0)
                ` Est-ce que la place libre est suffisante (plus 32K) ?
            Si ($vlLibre>=($1+32768))
                    ` Si oui, vérifier que le volume n'est pas verrouillé...
                $vaNomDoc:=$taVolumes{$vlVolume}+Caractere(Symbole séparateur )
                                                    "XYZ"+Chaine(Hasard)+".TXT"
                $vhDocRef:=Creer document($vaNomDoc)
                Si (OK=1)
                    FERMER DOCUMENT($vhDocRef)
                    SUPPRIMER DOCUMENT($vaNomDoc)
                        ` Si tout est ok, retourner le nom du volume
                    $0:=$taVolumes{$vlVolume}
                    $vlVolume:=$vlNbVolumes+1
                Fin de si
```

```
            Fin de si
        Fin de si
    Fin de boucle
Fin de si
APPELER SUR ERREUR("")
```

Lorsque cette méthode projet est ajoutée à votre application, vous pouvez écrire :

```
$vaVolume:=Chercher volume pour place (100*1024*1024)
Si ($vaVolume#"")
    ` Continuer
Sinon
    ALERTE("Un volume avec au moins 100 Mo d'espace libre est nécessaire !")
Fin de si
```

Référence

LISTE DES VOLUMES.

RESOUDRE ALIAS (cheminAlias; cheminCible)

Paramètre	Type		Description
cheminAlias	Alpha	→	Nom ou chemin d'accès complet de l'alias/ du raccourci
cheminCible	Alpha	←	Nom ou chemin d'accès complet de la cible de l'alias/du raccourci

Description

La commande RESOUDRE ALIAS retourne le chemin d'accès complet du fichier ou dossier cible d'un alias (appelé "raccourci" sous Windows).

Vous passez dans cheminAlias le nom ou le chemin d'accès complet de l'alias.

Note : Sous Windows, les raccourcis sont des fichiers dont l'extension est ".LNK". Si vous ne passez pas cette extension, la commande l'ajoute automatiquement.

Après l'exécution de la commande, la variable cheminCible contient le chemin d'accès complet du fichier ou dossier cible de l'alias et la variable système OK prend la valeur 1. Si le chemin passé dans cheminAlias correspond à un fichier et non à un alias, cheminCible retourne le chemin d'accès du fichier et la variable système OK prend la valeur 0.

Référence

CREER ALIAS.

Variables et ensembles système

Si cheminAlias désigne bien un alias/raccourci, la variable système OK prend la valeur 1. Si cheminAlias désigne un fichier standard, la variable système OK prend la valeur 0.

Selectionner document

version 11 (Modifiée)

Selectionner document (répertoire; typesFichiers; titre; options{; sélectionnés}) → Chaîne

Paramètre	Type		Description
répertoire	Texte | Entier long	→	• Chemin d'accès du répertoire à afficher par défaut dans la boîte de dialogue de sélection, ou • Chaîne vide pour afficher le dossier d'utilisateur par défaut ("Mes documents" sous Windows, "Documents" sous Mac OS), ou • Numéro de chemin d'accès mémorisé
typesFichiers	Texte	→	Liste des types de documents à filtrer, ou "*" pour ne pas filtrer les documents
titre	Texte	→	Titre de la boîte de dialogue de sélection
options	Entier long	→	Option(s) de sélection
sélectionnés	Tableau Texte	←	Tableau contenant la liste des chemins d'accès + les noms des fichiers sélectionnés
Résultat	Chaîne	←	Nom du fichier sélectionné (premier fichier de la liste en cas de sélection multiple)

Description

La commande Selectionner document affiche une boîte de dialogue standard d'ouverture de documents, permettant à l'utilisateur de désigner un ou plusieurs fichier(s), et retourne le nom et/ou le chemin d'accès complet du ou des fichier(s) sélectionné(s).

Le paramètre répertoire indique le dossier dont le contenu doit être affiché initialement dans la boîte de dialogue d'ouverture de documents. Vous pouvez passer trois types de valeurs :
• un texte contenant le chemin d'accès complet du répertoire à afficher.
• une chaîne vide ("") pour afficher le dossier d'utilisateur par défaut du système ("Mes documents" sous Windows, "Documents" sous Mac OS).
• un numéro de chemin d'accès mémorisé (de 1 à 32000) pour afficher le dossier associé.
Avec ce principe, vous pouvez conserver en mémoire le chemin d'accès du dossier ouvert au moment où l'utilisateur a cliqué sur le bouton de sélection, c'est-à-dire le dossier choisi par l'utilisateur. Lors du premier appel d'un numéro arbitraire (par exemple 5), la commande affiche le dossier d'utilisateur par défaut du système (équivaut à passer une chaîne vide). L'utilisateur pourra alors éventuellement naviguer parmi les dossiers de son disque dur. Au moment où il cliquera sur le bouton de sélection, le chemin d'accès sera mémorisé et associé au numéro 5. Lors des appels suivants du numéro 5, le chemin d'accès mémorisé sera utilisé par défaut. En cas de sélection d'un nouvel emplacement, le chemin numéro 5 sera mis à jour, et ainsi de suite.

Ce mécanisme vous permet de mémoriser jusqu'à 32000 chemins d'accès. Sous Windows, chaque chemin est conservé durant la session uniquement. Sous Mac OS, les chemins sont conservés par le système, ils restent mémorisés d'une session à l'autre.

Note : Ce mécanisme est identique à celui utilisé par la commande Selectionner dossier. Les numéros de chemins d'accès mémorisés sont partagés entre les deux commandes.

Passez dans le paramètre typeFichiers le ou les type(s) de fichier(s) pouvant être sélectionnés dans la boîte de dialogue d'ouverture. Vous pouvez passer une liste de plusieurs types séparés par un ; (point virgule). Pour chaque type défini, une ligne sera ajoutée dans le menu de choix de type de la boîte de dialogue.
Sous Mac OS, vous pouvez passer soit un type Mac OS classique (TEXT, APPL, etc.), soit un type UTI (*Uniform Type Identifier*). Les types UTIs ont été définis par Apple afin de répondre aux besoins d'uniformisation des types de fichiers. Par exemple, "public.text" est le type UTI des fichiers de type texte. Pour plus d'informations sur les UTIs, reportez-vous à l'adresse http://developer.apple.com/documentation/Carbon/Conceptual/understanding_utis/index.html (documentation en anglais).
Sous Windows, vous pouvez également passer un type de fichier classique Mac OS — 4D effectue la correspondance en interne — ou l'extension des fichiers (.txt, .exe, etc.). A noter que sous Windows, l'utilisateur aura la possibilité de "forcer" l'affichage de tous les types de fichiers en saisissant ***.*** dans la boîte de dialogue. Toutefois dans ce cas, 4D effectuera une vérification supplémentaire des types des fichiers sélectionnés : si l'utilisateur sélectionne un type de fichier non autorisé, la commande retourne une erreur.
Si vous ne souhaitez pas restreindre les fichiers affichés à un ou plusieurs types, passez la chaîne "*" (étoile) ou ".*" dans typeFichiers.

Passez dans le paramètre titre le libellé devant apparaître dans la boîte de dialogue. Par défaut, si vous passez une chaîne vide, le libellé "Ouvrir" est affiché.

Le paramètre options permet de spécifier les fonctions avancées autorisées dans la boîte de dialogue d'ouverture. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème "Documents système" :

Constante	Type	Valeur
Fichiers multiples	Entier long	1
Ouverture progiciel	Entier long	2
Sélection progiciel	Entier long	4
Sélection alias	Entier long	8
Utiliser fenêtre feuille	Entier long	16

Vous pouvez passer une constante ou une combinaison de constantes.
• Fichiers multiples : autoriser la sélection simultanée de plusieurs fichiers à l'aide des combinaisons **Maj+clic** (sélection contiguë) et **Ctrl+clic** (Windows) ou **Commande+clic** (Mac OS). Dans ce cas, le paramètre sélectionnés, s'il est passé, contient la liste de tous les fichiers sélectionnés. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas la sélection de plusieurs fichiers.

• Ouverture progiciel (Mac OS uniquement) : autoriser l'ouverture des progiciels (packages) en tant que dossiers et donc la visualisation/sélection de leur contenu. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas l'ouverture des progiciels.
• Sélection progiciel (Mac OS uniquement) : autoriser la sélection de progiciels (packages) en tant qu'entités. Par défaut, si cette constante n'est pas utilisée, la commande ne permet pas de sélectionner les progiciels en tant que tels.
• Sélection alias : autoriser la sélection de raccourcis (Windows) ou d'alias (Mac OS) en tant que documents. Par défaut, si cette constante n'est pas utilisée, la commande n'autorise pas la sélection de raccourcis ou d'alias en tant que tels. Si l'utilisateur sélectionne ce type de document, la commande retourne le chemin de l'élément cible. Lorsque vous passez la constante, la commande retourne le chemin de l'alias ou du raccourci lui-même.
• Utiliser fenêtre feuille (Mac OS uniquement) : afficher la boîte de dialogue de sélection sous forme de fenêtre feuille (cette option est ignorée sous Windows).
Les fenêtres feuilles sont des fenêtres spécifiques de l'interface Mac OS X, bénéficiant notamment d'une animation graphique (pour plus d'informations, reportez-vous à la section Types de fenêtres). Par défaut, si cette constante n'est pas utilisée, la commande affiche une boîte de dialogue standard.

Si vous ne souhaitez pas utiliser d'option, passez 0 dans le paramètre options.

Le paramètre facultatif sélectionnés permet de récupérer le chemin d'accès complet (chemin d'accès + nom) de chaque fichier sélectionné par l'utilisateur. La commande crée, dimensionne et remplit le tableau en fonction de la sélection de l'utilisateur. Ce paramètre est utile lorsque l'option Fichiers multiples est utilisée, ou lorsque vous souhaitez connaître le chemin d'accès du fichier sélectionné (il suffit dans ce cas de soustraire de la valeur du tableau le nom du fichier retourné par la commande). Si aucun fichier n'a été sélectionné, le tableau est retourné vide.

Note : Sous Mac OS, un progiciel sélectionné est considéré comme un dossier. Le chemin d'accès retourné dans le tableau sélectionnés comporte un caractère ":" final. Par exemple : Disque:Applications:4D:4D v11.4:FR:4D Volume Desktop.app:

La commande retourne le nom (nom+extension sous Windows) du fichier sélectionné. Si plusieurs fichiers ont été sélectionnés, la commande retourne le nom du premier fichier de la liste des fichiers sélectionnés. La liste des fichiers peut être récupérée dans le paramètre sélectionnés. Si aucun fichier n'a été sélectionné, la commande retourne une chaîne vide.

Exemple

Cet exemple permet de désigner un fichier de données 4D :

```
C_ENTIER LONG($platForm)
PROPRIETES PLATE FORME($platForm)
Si ($platForm=Windows )
    $DocType:=".4DD"
Sinon
    $DocType:="com.4d.4d.data-file"   `Type UTI
Fin de si
```

```
$Options:=Sélection alias +Ouverture progiciel +Utiliser fenêtre feuille
$Doc:=Selectionner document("";$DocType;"Sélectionner le fichier de données";
                                                                    $Options)
```

Référence

Ouvrir document, Selectionner dossier.

Variables et ensembles système

Si la commande a été correctement exécutée et qu'un document valide a été sélectionné, la variable système *OK* prend la valeur 1 et la variable système *Document* contient le chemin d'accès complet du fichier sélectionné.
Si aucun fichier n'a été sélectionné (par exemple si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue d'ouverture), la variable système *OK* prend la valeur 0 et la variable système *Document* est vide.

Selectionner dossier ({message}{; }{répertoire}) → Alpha

Paramètre	Type		Description
message	Chaîne	→	Titre de la fenêtre de sélection
répertoire	Chaîne | Entier long	→	• Chemin d'accès du répertoire par défaut ou • Chaîne vide pour afficher le dossier d'utilisateur par défaut ("Mes documents" sous Windows, "Documents" sous Mac OS), ou • Numéro de chemin d'accès mémorisé
Résultat	Alpha	←	Chemin d'accès au dossier sélectionné

Description

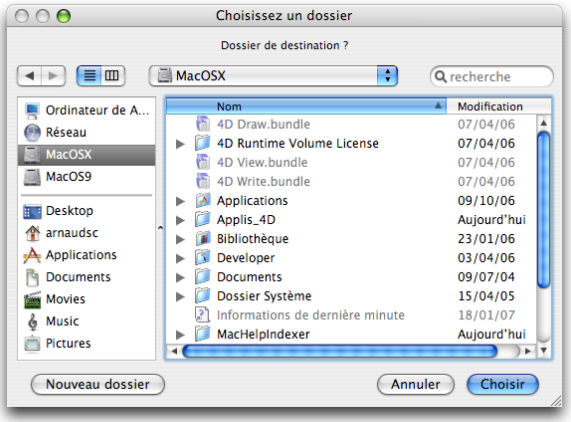
La commande Selectionner dossier affiche une boîte de dialogue permettant de désigner manuellement un dossier, et de récupérer en retour de fonction le chemin d'accès complet au dossier sélectionné. Le paramètre facultatif répertoire vous permet de désigner un emplacement de dossier qui sera affiché initialement dans la boîte de dialogue de sélection de dossier.

Note : Cette commande ne modifie pas le dossier courant de l'application 4D.

La commande Selectionner dossier affiche une boîte de dialogue standard de navigation à travers les volumes et les dossiers du poste.
Le paramètre optionnel message permet d'afficher une ligne d'information dans la boîte de dialogue (dans notre exemple, message a pour valeur "Dossier de destination ?").

• *Windows* :

• *Mac OS* :



Vous pouvez utiliser le paramètre répertoire pour proposer un emplacement de dossier par défaut dans la boîte de dialogue de sélection de dossier. Vous pouvez passer dans ce paramètre trois types de valeurs :
• un chemin d'accès de dossier valide utilisant la syntaxe de la plate-forme courante.
• une chaîne vide ("") pour afficher le dossier d'utilisateur par défaut du système ("Mes documents" sous Windows, "Documents" sous Mac OS).
• un numéro de chemin d'accès mémorisé (de 1 à 32000) pour afficher le dossier associé.
Avec ce principe, vous pouvez conserver en mémoire le chemin d'accès du dossier ouvert au moment où l'utilisateur a cliqué sur le bouton de sélection, c'est-à-dire le dossier choisi par l'utilisateur. Lors du premier appel d'un numéro arbitraire (par exemple 5), la commande affiche le dossier d'utilisateur par défaut du système (équivaut à passer une chaîne vide). L'utilisateur peut alors éventuellement naviguer parmi les dossiers de son disque dur. Au moment où il clique sur le bouton de sélection, le chemin d'accès est mémorisé et associé au numéro 5. Lors des appels suivants du numéro 5, le chemin d'accès mémorisé sera utilisé par défaut. En cas de sélection d'un nouvel emplacement, le chemin numéro 5 sera mis à jour, et ainsi de suite.
Ce mécanisme vous permet de mémoriser jusqu'à 32000 chemins d'accès. Sous Windows, chaque chemin est conservé durant la session uniquement. Sous Mac OS, les chemins restent mémorisés d'une session à l'autre. Si le chemin d'accès est incorrect, le paramètre cheminDéfaut est ignoré.

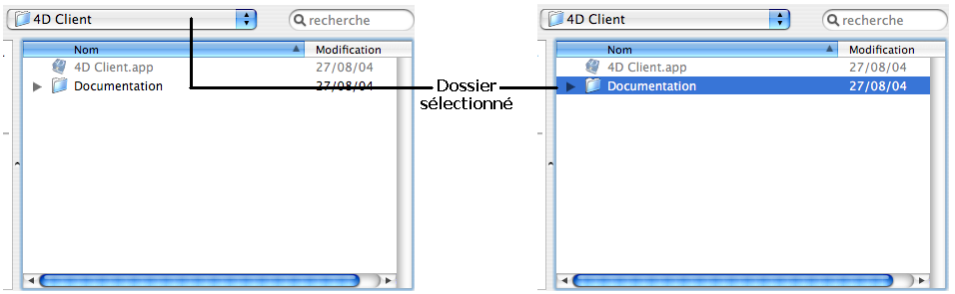
Note : Ce mécanisme est identique à celui utilisé par la commande Selectionner document. Les numéros de chemins d'accès mémorisés sont partagés entre les deux commandes.

L'utilisateur sélectionne un dossier en cliquant sur le bouton **OK** (Windows) ou **Sélectionner** (Mac OS). Le chemin d'accès au dossier choisi est alors retourné par la fonction.

• Sous Windows, la chaîne retournée est du type :
"C:\Dossier1\Dossier2\DossierSélectionné\"

• Sous Mac OS, la chaîne retournée est du type :
"Disque:Dossier1:Dossier2:DossierSélectionné:"

Note Mac OS : Sous Mac OS, selon que le nom du dossier est sélectionné ou non dans la boîte de dialogue, le chemin retourné est différent.



4D Server : Cette fonction permet de visualiser les volumes connectés aux postes clients. Il n'est pas possible de l'appeler depuis une procédure stockée.

Si l'utilisateur clique sur le bouton de sélection, la variable système *OK* prend la valeur 1. Si l'utilisateur clique sur le bouton d'annulation, *OK* prend la valeur 0 et la fonction retourne une chaîne vide.

Note : Sous Windows, si l'utilisateur a sélectionné certains éléments incorrects tels que "Poste de travail", "Corbeille", etc., la variable système *OK* prend la valeur 0, même si la boîte de dialogue est validée.

Exemple

L'exemple suivant permet de sélectionner le dossier dans lequel toutes les images de la bibliothèque d'images seront enregistrées :

```
$DossierImages:=Selectionner dossier("Sélectionnez un dossier pour vos images.")
LISTE IMAGES DANS BIBLIOTHEQUE (pictRefs;pictNoms)
Boucle ($n;1;Taille tableau(pictNames))
    $vRef:=Creer document($DossierImages+pictNoms{$n};"PICT")
    Si (OK=1)
        LIRE IMAGE DANS BIBLIOTHEQUE(pictRefs{$n};$vPictSauvegarde)
        ENREGISTRER IMAGE($vRef;$vPictSauvegarde)
        FERMER DOCUMENT($vRef)
    Fin de si
Fin de boucle
```

Référence

CREER DOSSIER, LISTE DES DOSSIERS, Selectionner document.

Variables et ensembles système

Si l'utilisateur clique sur le bouton de sélection, la variable système *OK* prend la valeur 1. Si l'utilisateur clique sur le bouton d'annulation, *OK* prend la valeur 0.

SUPPRIMER DOCUMENT (document)

Paramètre	Type		Description
document	Alpha	→	Nom de document ou Chemin d'accès complet au document

Description

SUPPRIMER DOCUMENT supprime le document dont vous avez passé le nom dans document.

Si le nom du document ou le chemin d'accès saisi est incorrect, une erreur est générée. Il en va de même si vous tentez de supprimer un document ouvert.

SUPPRIMER DOCUMENT n'accepte pas de chaîne vide dans le paramètre document. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichiers ne s'affiche pas et une erreur est générée.

ATTENTION : SUPPRIMER DOCUMENT peut supprimer tout fichier disque, y compris des fichiers créés par d'autres applications ou les applications elles-mêmes. La commande SUPPRIMER DOCUMENT doit donc être utilisée avec précaution. La suppression d'un document est une opération définitive et irréversible.

Exemples

(1) L'exemple suivant supprime le document appelé Note :

```
SUPPRIMER DOCUMENT ("Note") ` Suppression du document
```

(2) Reportez-vous à l'exemple de la commande AJOUTER DONNEES AU CONTENEUR.

Variables et ensembles système

La suppression d'un document met la variable système OK à 1. Si SUPPRIMER DOCUMENT ne peut pas supprimer le document, la variable système OK prend la valeur 0.

SUPPRIMER DOSSIER (dossier)

Paramètre	Type		Description
dossier	Alpha	→	Nom ou chemin d'accès complet du dossier à supprimer

Description

La commande SUPPRIMER DOSSIER supprime le dossier dont vous avez passé le nom ou le chemin d'accès complet dans dossier.

Seuls les dossiers vides peuvent être supprimés par cette commande.
• Si vous tentez de supprimer un dossier contenant des éléments, l'erreur -47 (Tentative de suppression d'un dossier non vide) est générée.
• Si vous passez dans dossier le chemin d'accès d'un fichier, ou une chaîne vide, ou encore le chemin d'accès d'un dossier inexistant, la commande ne fait rien et génère l'erreur -43 (Fichier non trouvé).

Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

Référence

SUPPRIMER DOCUMENT.

Taille document

version 11.3 (Modifiée)

Taille document (document{; *}) → Réel

Paramètre	Type		Description
document	DocRef | Alpha	→	Numéro de référence de document ou Nom de document
*		→	Mac OS uniquement : - si omis, taille de la data fork - si passé, taille de la resource fork
Résultat	Réel	←	Taille (en octets) de document

Description

La commande Taille document retourne la taille, exprimée en octets, d'un document.

Si le document est ouvert, passez son numéro de référence dans document.
Si le document n'est pas ouvert, passez son nom ou son chemin d'accès dans document.

Sous Mac OS, si vous ne passez pas le paramètre optionnel *, la taille de la data fork est retournée. Si vous passez le paramètre optionnel *, la taille de la resource fork est retournée.

Référence

CHANGER POSITION DANS DOCUMENT, CHANGER TAILLE DOCUMENT, Position dans document.

Tester chemin acces (cheminAccès) → Numérique

Paramètre	Type		Description
cheminAccès	Alpha	→	Chemin d'accès à un dossier ou un document
Résultat	Numérique	←	1= cheminAccès est un document existant 0 = cheminAccès est un dossier existant <0 = chemin d'accès invalide, code d'erreur du gestionnaire de fichiers du système

Description

La fonction Tester chemin acces vérifie si le document ou le dossier dont vous avez passé le chemin d'accès et le nom dans cheminAccès est présent sur le disque. Vous pouvez passer un chemin d'accès relatif ou absolu, exprimé dans la syntaxe du système courant.

Si un document est trouvé, Tester chemin acces retourne 1. Si un dossier est trouvé, Tester chemin acces retourne 0.

4D propose les constantes prédéfinies suivantes :

Constante	Type	Valeur
Est un document	Entier long	1
Est un répertoire	Entier long	0

Si aucun document ou dossier n'est trouvé, Tester chemin acces retourne une valeur négative (par exemple -43 pour "Fichier non trouvé").

Exemple

L'exemple suivant teste la présence du document “Journal” dans le dossier de la base et le crée s'il n'existe pas :

```
Si (Tester chemin acces("Journal") # Est un document)
    $vhDocRef:=Creer document("Journal")
    Si (OK=1)
        FERMER DOCUMENT($vhDocRef)
    Fin de si
Fin de si
```

Référence

Creer document, CREER DOSSIER.

Type document

Type document (document) → Alpha

Paramètre	Type		Description
document	Alpha	→	Nom de document ou Chemin d'accès complet à un document
Résultat	Alpha	←	Extension de fichier Windows (chaîne de 1 à N caractères) ou type de fichier Mac OS (chaîne de 4 caractères)

Description

La commande Type document retourne le type du document dont vous avez passé le nom ou le chemin d'accès dans document.

Sous Windows, Type document retourne l'extension de fichier du document (par exemple 'DOC' pour un document Microsoft Word, 'EXE' pour un fichier exécutable, etc.) ou le type de document Mac OS (4 caractères) correspondant si ce dernier a été associé à une extension Windows équivalente par 4D (par exemple 'TEXT' pour l'extension 'TXT') ou par un appel antérieur à la commande ASSOCIER TYPES FICHIER.

Sous Mac OS, Type document retourne, s'il est défini, le type de fichier Mac OS (4 caractères) du document (par exemple 'TEXT' pour un document de type Texte, 'APPL' pour une application double-cliquable, etc.).

Note de compatibilité : L'utilisation des types de fichiers Mac OS est obsolète sous Mac OS X. Comme sous Windows, l'identification des fichiers est désormais basée sur le suffixe de leur nom (cf. section Présentation des documents système). Par compatibilité, il reste cependant possible de lire le type Mac OS des documents lorsqu'il a été défini.

Référence

ASSOCIER TYPES FICHIER, CHANGER TYPE DOCUMENT, Createur document, PROPRIETES DOCUMENT.

13

Enregistrements

A propos des numéros d'enregistrements

Dans 4D, trois numéros sont associés à un enregistrement :
- Numéro d'enregistrement,
- Numéro dans la sélection,
- Numéro automatique.

Numéro d'enregistrement
Le numéro d'enregistrement est le numéro physique/absolu de l'enregistrement. Ce numéro est automatiquement assigné à chaque nouvel enregistrement et reste le même jusqu'à ce que cet enregistrement soit détruit. Les enregistrements commencent au numéro zéro (0).
Les numéros d'enregistrements ne sont pas uniques car les numéros des enregistrements détruits sont réutilisés pour de nouveaux enregistrements. Ces numéros sont également modifiés lorsque la base est réparée ou compactée.

Numéro dans la sélection
Le numéro dans la sélection est la position de l'enregistrement dans la sélection courante. Ce numéro dépend de la sélection courante. Si la sélection est modifiée ou triée, ce numéro change aussi probablement. La numérotation dans une sélection courante commence à un (1).

Numéro automatique
Le numéro automatique est un numéro unique, non répétable, qui peut être assigné à un champ dans un enregistrement (via la propriété **Incrémentation auto**, l'attribut SQL AUTO_INCREMENT ou la commande Numerotation automatique). Il n'est pas automatiquement stocké à chaque enregistrement. Il démarre par défaut à 1 et est incrémenté à chaque création d'un nouvel enregistrement. A la différence des numéros d'enregistrements, un numéro automatique n'est pas réutilisé lorsque l'enregistrement est détruit, ou lorsque la base est compactée ou réparée.
Ces numéros fournissent un moyen d'attribuer un numéro d'identification unique à chaque enregistrement. Si un numéro automatique est incrémenté pendant une transaction, ce numéro n'est pas décrémenté si la transaction est annulée.

Note : 4D n'effectue pas de contrôle lorsque vous modifiez le compteur interne des numéros automatiques d'une table à l'aide de la commande FIXER PARAMETRE BASE. Si vous décrémentez ce compteur, les nouveaux enregistrements créés pourront avoir des numéros ayant déjà été attribués.

Exemples de numéros d'enregistrements

Les tableaux suivants comparent le fonctionnement des différents numéros d'enregistrements. Chaque ligne de tableau représente les informations d'un enregistrement. L'ordre des lignes est celui dans lequel les enregistrements seraient affichés dans un formulaire sortie.

• **Colonne des Données** : Les valeurs d'un champ dans chaque enregistrement. Elle contient le nom d'une personne.
• **Colonne de Numéro d'enregistrement (N° Enrg)** : C'est le numéro absolu de l'enregistrement et qui est retourné par la fonction Numero enregistrement.
• **Colonne de Numéro dans la sélection (N° Sélection)** : C'est le numéro de position dans la sélection courante, qui est retourné par la fonction Numero dans selection.
• **Colonne de Numéro automatique (N° Auto)**: C'est le numéro unique de l'enregistrement, qui est retourné par la fonction Numerotation automatique. Ce numéro est stocké dans un champ.

Après saisie des enregistrements :
Le premier tableau présente des enregistrements qui viennent d'être saisis.
• l'ordre des enregistrements par défaut est le numéro d'enregistrement.
• Le numéro d'enregistrement commence à 0.
• Le numéro dans la sélection et le numéro automatique commencent à 1.

Données	N° Enrg	N° Sélection	N° Auto
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Sam	3	4	4
Lisa	4	5	5

Note: Les enregistrements restent dans l'ordre par défaut après l'appel de toute commande qui modifie la sélection sans la réordonner, comme par exemple la commande de menu **Tout montrer** en mode Développement ou après l'exécution de la commande TOUT SELECTIONNER.

Après un tri des enregistrements :
La première partie du tableau présente les enregistrements triés par noms.
• Le numéro d'enregistrement reste associé à l'enregistrement.
• Le numéro dans la sélection reflète la nouvelle position de l'enregistrement dans la sélection triée.
• Le numéro automatique ne change jamais puisqu'il est assigné à la création de chaque enregistrement et stocké avec lui.

Données	N° Enrg	N° Sélection	N° Auto
Lisa	4	1	5
Sabra	2	2	3
Sam	3	3	4
Terri	1	4	2
Tess	0	5	1

Après la suppression d'un enregistrement :
Voici le tableau après la destruction de l'enregistrement de Sam.
• Seuls les numéros dans la sélection ont changé (les numéros dans la sélection reflètent l'ordre d'affichage des enregistrements).

Données	N° Enrg	N° Sélection	N° Auto
Lisa	4	1	5
Sabra	2	2	3
Terri	1	3	2
Tess	0	4	1

Après l'ajout d'un enregistrement :
Voici le tableau après l'ajout de l'enregistrement Liz.
• Un nouvel enregistrement est ajouté à la fin de la sélection courante.
• Le numéro d'enregistrement de Sam est réutilisé pour le nouvel enregistrement.
• Le numéro automatique a été incrémenté de 1.

Données	N° Enrg	N° Sélection	N° Auto
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Lisa	4	4	5
Liz	3	5	6

Après un changement de sélection et un tri :
Le tableau qui suit montre les enregistrements après réduction de la sélection à trois enregistrements qui sont ensuite triés.
• Seuls le numéro dans la sélection change.

Données	N° Enrg	N° Sélection	N° Auto
Sabra	2	1	3
Liz	3	2	6
Terri	1	3	2

Référence

Numero dans selection, Numero enregistrement, Numerotation automatique.

Les commandes EMPILER ENREGISTREMENT et DEPILER ENREGISTREMENT vous permettent de poser (“empiler”) des enregistrements sur le dessus de la pile des enregistrements, et de les enlever (“dépiler”) de la pile.

Chaque process dispose de sa propre pile d'enregistrements pour chaque table. 4D gère pour vous les piles d'enregistrements. Chaque pile d'enregistrements est du type LIFO ("Last-In-First-Out", ce qui peut se traduire par "dernier-entré-premier-sorti"). La capacité de la pile dépend de la mémoire.

Les commandes EMPILER ENREGISTREMENT et DEPILER ENREGISTREMENT doivent être utilisées avec prudence. Chaque enregistrement empilé utilise une partie de la mémoire disponible. Empiler trop d'enregistrements peut causer l'apparition d'un message du type "mémoire insuffisante" ou une pile pleine.

4D efface de la pile les enregistrements "dépilés" quand vous retournez au menu à la fin de l'exécution de la méthode.

EMPILER ENREGISTREMENT et DEPILER ENREGISTREMENT sont utiles lorsque par exemple, en cours de saisie, vous voulez examiner des enregistrements se trouvant dans la même table que celle que vous êtes en train d'utiliser. Pour cela, vous empilez votre enregistrement, cherchez et examinez les enregistrements dans la table (vous copiez des champs dans des variables, par exemple), et finalement vous dépilez l'enregistrement pour le restaurer.

Note pour les utilisateurs de la version 5 de 4D : Quand vous saisissez un enregistrement, si vous devez vérifier l'unicité d'une valeur sur plusieurs champs, utilisez la nouvelle commande FIXER DESTINATION RECHERCHE. Cela vous évitera les appels à EMPILER ENREGISTREMENT et DEPILER ENREGISTREMENT que vous deviez effectuer avant d'utiliser CHERCHER, afin de préserver les données saisies dans l'enregistrement courant. FIXER DESTINATION RECHERCHE permet d'exécuter une recherche qui ne change pas la sélection ni l'enregistrement courants.

Référence

DEPILER ENREGISTREMENT, EMPILER ENREGISTREMENT, FIXER DESTINATION RECHERCHE.

AFFICHER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle afficher l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

AFFICHER ENREGISTREMENT affiche l'enregistrement courant de table dans le formulaire entrée courant. L'enregistrement reste affiché jusqu'à ce qu'un événement provoque un redessinement de la fenêtre. Cet événement peut être l'exécution d'un AJOUTER ENREGISTREMENT, le retour au formulaire entrée ou à la barre de menus. AFFICHER ENREGISTREMENT ne fait rien s'il n'y a pas d'enregistrement courant.

AFFICHER ENREGISTREMENT est souvent utilisé pour afficher des messages de progression personnalisés. Cette commande peut également servir à générer un "slide show" automatique.

Si une méthode formulaire existe, un événement Sur chargement est généré.

Exemple

L'exemple suivant affiche une série d'enregistrements sous forme de slide show :

```
TOUT SELECTIONNER([Démo]) ` Sélection de tous les enregistrements
FORMULAIRE ENTREE ([Démo]; "Affichage") ` Désignation du formulaire à utiliser
Boucle ($i; 1; Enregistrements trouves([Démo])) ` Boucle sur tous les enregistrements
   AFFICHER ENREGISTREMENT([Démo]) ` Afficher un enregistrement
   ENDORMIR PROCESS (Numero du process courant; 180) ` 3 secondes de pause
   ENREGISTREMENT SUIVANT([Démo]) ` Passage à l'enregistrement suivant
Fin de boucle
```

Référence

MESSAGE.

ALLER A ENREGISTREMENT ({table; }enregistrement)

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement de destination ou Table par défaut si ce paramètre est omis
enregistrement	Numérique	→	Numéro renvoyé par Numero enregistrement

Description

ALLER A ENREGISTREMENT sélectionne l'enregistrement courant de table. Le paramètre enregistrement est le numéro renvoyé par la fonction Numero enregistrement. Après l'exécution de cette commande, l'enregistrement est le seul de la sélection courante.

Si enregistrement est inférieur au plus petit numéro d'enregistrement ou supérieur au plus grand numéro d'enregistrement de la base, 4D génère un message d'erreur indiquant que le numéro est hors intervalle. Si enregistrement est égal au numéro d'un enregistrement supprimé, 4D retourne l'erreur -10503 et la sélection courante devient vide.

Exemple

Référez-vous à l'exemple de la commande Numero enregistrement.

Référence

A propos des numéros d'enregistrements, Numero enregistrement.

CREER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table dans laquelle créer un enregistrement ou Table par défaut si ce paramètre est omis

Description

CREER ENREGISTREMENT crée un nouvel enregistrement vide pour la table table, mais ne l'affiche pas à l'écran. Vous devez utiliser la commande AJOUTER ENREGISTREMENT pour créer un nouvel enregistrement et l'afficher dans un formulaire entrée.

Utilisez CREER ENREGISTREMENT plutôt que AJOUTER ENREGISTREMENT lorsque les valeurs de l'enregistrement sont entrées par programmation. Le nouvel enregistrement devient l'enregistrement courant mais la sélection courante n'est pas modifiée.

L'enregistrement est créé uniquement en mémoire et doit être sauvegardé à l'aide de STOCKER ENREGISTREMENT. Si vous changez d'enregistrement courant (par exemple à la suite d'une recherche) avant la sauvegarde, l'enregistrement créé est perdu.

Exemple

L'exemple suivant archive les enregistrements datant de plus de 30 jours. Cette opération est réalisée par la création d'enregistrements dans une table d'archive. Une fois l'opération terminée, les enregistrements archivés sont supprimés de la table [Comptes] :

```
    ` Recherche des enregistrements datant de plus de 30 jours
CHERCHER ([Comptes]; [Comptes]Saisie < (Date du jour – 30))
    ` Boucle une fois par enregistrement
Boucle ($vlRecord; 1; Enregistrements trouves([Comptes]))
        ` Création d'un nouvel enregistrement d'archive
    CREER ENREGISTREMENT ([Archives])
    [Archive]Numéro := [Comptes]Number ` Copie des champs dans l'archive
    [Archive]Saisie := [Comptes]Saisie
    [Archive]Montant := [Comptes]Montant
```

```
        STOCKER ENREGISTREMENT ([Archive]) ` Sauvegarde de l'enregistrement d'archive
            ` Passage à l'enregistrement de compte suivant
        ENREGISTREMENT SUIVANT ([Comptes])
    Fin de boucle
    SUPPRIMER SELECTION([Comptes]) ` Suppression des enregistrements
```

Référence

AJOUTER ENREGISTREMENT, MODIFIER ENREGISTREMENT, STOCKER ENREGISTREMENT.

DEPILER ENREGISTREMENT

DEPILER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle dépiler l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

DEPILER ENREGISTREMENT charge le premier enregistrement de la pile d'enregistrements de table, et en fait l'enregistrement courant.

Si vous empilez un enregistrement puis créez une nouvelle sélection courante ne contenant plus l'enregistrement empilé, et enfin dépilez l'enregistrement, vous obtenez la situation dans laquelle l'enregistrement courant ne se trouve pas dans la sélection courante. Si vous souhaitez faire de l'enregistrement empilé la sélection courante, utilisez la commande ENREGISTREMENT SELECTION.
Si vous utilisez une routine qui déplace le pointeur d'enregistrement courant avant de sauvegarder l'enregistrement, vous perdrez la copie empilée en mémoire.

Exemples

L'exemple suivant récupère l'enregistrement d'un client dans la pile :

```
DEPILER ENREGISTREMENT ([Clients]) ` Dépiler l'enregistrement
```

Référence

EMPILER ENREGISTREMENT, Utiliser la pile d'enregistrements.

DUPLIQUER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement à dupliquer ou Table par défaut si ce paramètre est omis

Description

DUPLIQUER ENREGISTREMENT duplique l'enregistrement courant de table. Ce nouvel enregistrement devient l'enregistrement courant. S'il n'y a pas d'enregistrement courant, DUPLIQUER ENREGISTREMENT ne fait rien. Appelez la commande STOCKER ENREGISTREMENT pour sauvegarder le nouvel enregistrement.

DUPLIQUER ENREGISTREMENT peut être exécuté pendant la saisie des données. Vous pouvez donc dupliquer l'enregistrement qui est affiché à l'écran. N'oubliez pas que vous devez d'abord appeler STOCKER ENREGISTREMENT si vous voulez sauvegarder les modifications apportées à l'enregistrement original.

Référence

STOCKER ENREGISTREMENT.

EMPILER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle empiler l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

EMPILER ENREGISTREMENT "empile" une copie de l'enregistrement courant de table (ainsi que ses sous-enregistrements, le cas échéant) dans la pile d'enregistrements de la table. EMPILER ENREGISTREMENT peut être exécuté avant qu'un enregistrement soit sauvegardé.

Si vous empilez un enregistrement non verrouillé, il sera verrouillé pour tous les autres process et utilisateurs jusqu'à ce que vous le "dépiliez" (c'est-à-dire que vous le déchargiez de la pile).

Exemple

L'exemple suivant empile l'enregistrement d'un client :

```
EMPILER ENREGISTREMENT ([Client]) ` Placer l'enregistrement du client dans la pile
```

Référence

DEPILER ENREGISTREMENT, Utiliser la pile d'enregistrements.

Enregistrement charge {(table)} → Booléen

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement à examiner ou Table par défaut si ce paramètre est omis
Résultat	Booléen	←	Vrai si l'enregistrement est chargé, Faux sinon

Description

La commande Enregistrement charge retourne Vrai si l'enregistrement courant de table est chargé dans le process en cours.

Exemple

Au lieu d'utiliser les actions automatiques "Enregistrement suivant" ou "Enregistrement précédent", vous voulez écrire dans les méthodes de boutons sans action des instructions spécifiant que le bouton "Suivant" affiche le début de la sélection si la fin de la sélection est atteinte et que le bouton "Précédent" affiche la fin de la sélection si le début est atteint.

```
    ` Méthode objet du bouton sans action "PRÉCÉDENT"
Si (Evenement formulaire=Sur clic)
    ENREGISTREMENT PRECEDENT([Groupe])
    Si (Non(Enregistrement charge([Groupe])))
        ALLER DANS SELECTION([Groupe];Enregistrements trouves([Groupe]))
                `Aller au dernier enregistrement de la sélection
    Fin de si
Fin de si

    ` Méthode objet du bouton sans action "SUIVANT"
Si (Evenement formulaire=Sur clic)
    ENREGISTREMENT SUIVANT([Groupe])
    Si (Non(Enregistrement charge([Groupe])))
            `Aller au premier enregistrement de la sélection
        ALLER DANS SELECTION([Groupe];1)
    Fin de si
Fin de si
```

Enregistrement modifie {(table)} → Booléen

Paramètre	Type		Description
table	Table	→	Table de laquelle tester si l'enregistrement courant a été modifié ou Table par défaut si paramètre omis
Résultat	Booléen	←	L'enregistrement a été modifié (Vrai) ou L'enregistrement n'a pas été modifié (Faux)

Description

Enregistrement modifie retourne Vrai si l'enregistrement courant de table a été modifié et non encore stocké. Sinon, elle retourne Faux. Cette fonction vous permet de déterminer rapidement s'il faut stocker l'enregistrement. Dans les formulaires entrée, vous pouvez effectuer le test avant d'aller à l'enregistrement suivant. Cette fonction retourne toujours Vrai pour un nouvel enregistrement.

Exemples

L'exemple suivant montre une utilisation typique de Enregistrement modifie :

```
Si (Enregistrement modifie ([Clients]))
    STOCKER ENREGISTREMENT ([Clients])
Fin de si
```

Référence

Ancien, Modifie, STOCKER ENREGISTREMENT.

Enregistrements dans table

version 5

Enregistrements dans table {(table)} → Numérique

Paramètre	Type		Description
table	Table	→	Table de laquelle retourner le nombre total d'enregistrements ou Table par défaut si ce paramètre est omis
Résultat	Numérique	←	Nombre total d'enregistrements dans table

Description

Enregistrements dans table retourne le nombre total d'enregistrements que contient table. Par opposition, Enregistrements trouves retourne le nombre d'enregistrements de la sélection courante uniquement. Lorsque cette commande est utilisée dans une transaction, les enregistrements éventuellement créés pendant la transaction sont comptabilisés.

Exemple

L'exemple suivant affiche une alerte indiquant le nombre d'enregistrements de la table :

```
ALERTE ("Il y a " + Chaine (Enregistrements dans table ([Personnes])) +
                                                    " enregistrements  dans la table.")
```

Référence

Enregistrements trouves.

Nouvel enregistrement {(laTable)} → Booléen

Paramètre	Type		Description
laTable	Table	→	Table de l'enregistrement à examiner ou Table par défaut si ce paramètre est omis
Résultat	Booléen	←	Vrai si l'enregistrement est en cours de création, Faux sinon

Description

La commande Nouvel enregistrement retourne Vrai lorsque l'enregistrement courant de laTable est en cours de création et n'a pas encore été sauvegardé dans le process courant.

Note de compatibilité : Il est possible d'obtenir la même information avec la commande existante Numero enregistrement, en testant si elle retourne -3. Toutefois, il est vivement conseillé d'utiliser dans ce cas Nouvel enregistrement plutôt que Numero enregistrement. En effet, la commande Nouvel enregistrement assure une meilleure compatibilité avec les futures versions de 4D.

4D Server : Cette commande retourne un résultat différent dans le contexte de l'événement formulaire Sur validation suivant qu'elle est exécutée sur 4D en mode local ou 4D en mode distant. En mode local, la commande retourne Faux (l'enregistrement est considéré comme déjà créé). En mode distant, la commande retourne Vrai car dans ce cas, l'enregistrement est également déjà créé sur le serveur mais l'information n'a pas encore été envoyée au client.

Exemple

Les deux instructions suivantes sont identiques, la seconde est conseillée pour que le code reste compatible avec les prochaines versions de 4D :

```
Si (Numero enregistrement([Table])=-3) `Déconseillé
   ` ...
Fin de si

Si (Nouvel enregistrement([Table])) `Conseillé
   ` ...
Fin de si
```

Référence

Enregistrement modifie, Numero enregistrement.

Numero enregistrement {(laTable)} → Numérique

Paramètre	Type		Description
laTable	Table	→	Table de laquelle vous souhaitez obtenir le numéro de l'enregistrement courant ou Table par défaut si ce paramètre est omis
Résultat	Numérique	←	Numéro d'enregistrement courant

Description

Numero enregistrement retourne le numéro de l'enregistrement courant de laTable. S'il n'y a pas d'enregistrement courant, par exemple si le pointeur d'enregistrement se trouve avant ou après la sélection courante, Numero enregistrement retourne -1. S'il s'agit d'un nouvel enregistrement qui n'a pas encore été sauvegardé, Numero enregistrement retourne -3.

Les numéros d'enregistrements peuvent varier. Par exemple, les numéros des enregistrements supprimés sont réutilisés. Les numéros d'enregistrements changent aussi si vous compactez votre base.

4D Server : Cette commande retourne un résultat différent dans le contexte de l'événement formulaire Sur validation suivant qu'elle est exécutée sur 4D en mode local ou 4D en mode distant. En mode local, la commande retourne un numéro d'enregistrement (l'enregistrement est considéré comme déjà créé). En mode distant, la commande retourne -3 car dans ce cas, l'enregistrement est également déjà créé sur le serveur mais l'information n'a pas encore été envoyée au client.

Note : Il est fortement conseillé d'utiliser la commande Nouvel enregistrement pour vérifier si un enregistrement est en cours de création.

Exemple

L'exemple suivant sauvegarde le numéro d'enregistrement courant puis cherche dans la table si un autre enregistrement a la même valeur :

```
    ` Obtenir le numéro d'enregistrement
$NumEnreg := Numero enregistrement ([Personnes])
    ` Est-ce que quelqu'un d'autre a le même nom ?
CHERCHER ([Personnes]; [Personnes]Nom = [Personnes]Nom)
    ` Afficher une alerte avec le nombre de personnes qui ont le même nom
ALERTE ("Il existe " + Chaine (Enregistrements trouves ([Personnes]) + " personnes du
                                                                    même nom.")
    ` Retourner à l'enregistrement original
ALLER A ENREGISTREMENT ([Personnes]; $NumEnreg)
```

Référence

A propos des numéros d'enregistrements, ALLER A ENREGISTREMENT, Nouvel enregistrement, Numero dans selection, Numerotation automatique.

Numerotation automatique {(table)} → Numérique

Paramètre	Type		Description
table	Table	→	Table à numéroter automatiquement ou Table par défaut si ce paramètre est omis
Résultat	Numérique	←	Numéro automatique

Description

Numerotation automatique retourne le prochain numéro automatique de table. Ce numéro est unique pour chaque table. C'est une valeur qui ne se répète pas et qui est incrémentée à chaque enregistrement nouvellement créé dans la table. Par défaut, la numérotation commence à 1 ; vous pouvez toutefois modifier la numéroration automatique des enregistrements de table à l'aide de la commande FIXER PARAMETRE BASE.

Le numéro retourné par cette fonction pour la table est identique à celui généré si vous avez coché l'option **Incrémentation auto** dans l'Inspecteur de Structure pour un champde la table ou si vous fixez #N comme valeur par défaut pour un champ de la table dans un formulaire (référez-vous au manuel *Mode Développement* de 4D).

Note : L'incrémentation automatique peut également être définie via l'attribut SQL AUTO_INCREMENT.

La fonction Numerotation automatique est utile dans les cas suivants :
• Si la numérotation doit s'incrémenter d'un nombre supérieur à 1
• Si le numéro doit reprendre une partie d'un code

Pour stocker ce numéro à l'aide d'une méthode, il faut créer un champ de type Entier long dans la table et y affecter la numérotation automatique.
Si la numérotation doit débuter à une valeur différente de 1, ajoutez simplement la différence à la fonction Numerotation automatique. Par exemple, si le numéro doit commencer à 1000, vous pouvez utiliser la ligne de code suivante pour affecter le numéro :

```
[Table1]NumAuto := Numerotation automatique ([Table1]) + 999
```

Exemple

L'exemple suivant fait partie d'une méthode formulaire. Ces lignes de code testent s'il s'agit d'un nouvel enregistrement (si le numéro de facture est égal à une chaîne vide). Si l'enregistrement est nouveau, un numéro est affecté à la facture. Ce numéro de facture est construit avec deux informations : le numéro unique et le numéro de référence de l'utilisateur, qui était saisi à l'ouverture de la base. Le numéro unique est formaté en tant que chaîne avec une longueur de cinq caractères :

```
      ` S'il s'agit d'une nouvelle facture, créer un numéro de facture
Si ([Factures]NumFacture = "")
         ` Le numéro de facture est une chaîne qui se termine par le numéro de référence
         ` de l'utilisateur
   [Factures]NumFacture := Chaine (Numerotation automatique; "00000") +
                                                                  [Factures]Utilisateur
Fin de si
```

Référence

A propos des numéros d'enregistrements, Numero dans selection, Numero enregistrement.

STOCKER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement à stocker ou Table par défaut si ce paramètre est omis

Description

STOCKER ENREGISTREMENT sauvegarde l'enregistrement courant de table pour le process courant. S'il n'y a pas d'enregistrement courant, la commande est ignorée.

Vous pouvez utiliser STOCKER ENREGISTREMENT pour sauvegarder un enregistrement créé ou modifié par programmation. Lorsqu'un enregistrement a été modifié puis validé par un utilisateur dans un formulaire, il n'est pas nécessaire de le sauvegarder à l'aide de STOCKER ENREGISTREMENT. En revanche, un enregistrement modifié puis annulé par l'utilisateur peut malgré tout être sauvegardé avec STOCKER ENREGISTREMENT.

L'utilisation de STOCKER ENREGISTREMENT est nécessaire dans les cas suivants :
• Pour sauvegarder un enregistrement créé par les commandes CREER ENREGISTREMENT ou DUPLIQUER ENREGISTREMENT,
• Pour sauvegarder des données issues de la commande RECEVOIR ENREGISTREMENT,
• Pour sauvegarder un enregistrement modifié par une méthode,
• Pour sauvegarder un enregistrement contenant un sous-enregistrement ayant été créé ou modifié par la commande AJOUTER SOUS ENREGISTREMENT, CREER SOUS ENREGISTREMENT ou MODIFIER SOUS ENREGISTREMENT,
• Pendant la saisie de données, pour sauvegarder l'enregistrement affiché avant d'appeler une commande qui change l'enregistrement courant,
• Pendant la saisie de données, pour sauvegarder l'enregistrement courant.

Vous ne devez pas appeler STOCKER ENREGISTREMENT dans l'événement formulaire Sur validation d'un enregistrement qui a été validé, sinon l'enregistrement est sauvegardé deux fois.

Exemples

L'exemple suivant est une partie d'une méthode récupérant des enregistrements d'un fichier. Dans cette partie, les enregistrements sont reçus puis, si l'opération s'est correctement déroulée, sauvegardés :

```
      ` Réception de l'enregistrement à partir du disque
RECEVOIR ENREGISTREMENT ([Clients])
Si (OK = 1)  ` Si l'enregistrement a été correctement reçu…
   STOCKER ENREGISTREMENT ([Clients])  ` Le sauvegarder
Fin de si
```

Référence

CREER ENREGISTREMENT, Enregistrement verrouille, Présentation des triggers.

SUPPRIMER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle supprimer l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

SUPPRIMER ENREGISTREMENT supprime de table l'enregistrement courant du process en cours. S'il n'y a pas d'enregistrement courant pour table dans le process, SUPPRIMER ENREGISTREMENT ne fait rien. Dans un formulaire, vous pouvez créer un bouton 'Supprimer enregistrement' et lui assigner l'action automatique correspondante, plutôt que d'utiliser cette commande.

Note : Si l'enregistrement courant est déchargé de la mémoire avant l'appel à SUPPRIMER ENREGISTREMENT (par exemple suite à un LIBERER ENREGISTREMENT), la sélection courante de table est vide à l'issue de la suppression.

La suppression d'enregistrements est une opération définitive et ne peut être annulée.

Lorsqu'un enregistrement est supprimé, son numéro interne est réutilisé lors de la création de nouveaux enregistrements. Par conséquent, n'utilisez pas ces numéros comme identifiants de vos enregistrements si votre base permet la suppression d'enregistrements.

Exemple

L'exemple suivant permet de supprimer l'enregistrement d'un employé. La méthode demande à l'utilisateur le numéro de l'employé à supprimer, recherche l'enregistrement correspondant puis le supprime :

```
vCherch := Demander ("Numéro de l'employé à supprimer :")
    ` On récupère un numéro d'identification
Si (OK = 1)
    CHERCHER ([Employés]; [Employés]Numéro = vCherch)  ` Trouver l'employé
    SUPPRIMER ENREGISTREMENT ([Employés])  ` Suppression de l'enregistrement
Fin de si
```

Référence

Enregistrement verrouille, Présentation des triggers.

14

Enregistrements (verrouillage)

4D et 4D Server gèrent automatiquement les bases en empêchant les conflits entre les process ou entre les utilisateurs. Deux utilisateurs ou deux process ne peuvent modifier en même temps le même enregistrement ou le même objet. Le second utilisateur ou process peut toutefois accéder simultanément en "lecture seulement" à l'enregistrement ou à l'objet.

Vous devez utiliser les commandes multi-utilisateurs de cette section dans plusieurs cas :
• Modification d'enregistrements par programmation,
• Utilisation d'une interface utilisateur personnalisée pour les opérations multi-utilisateurs,
• Sauvegarde de modifications reliées entre elles dans une transaction.

Il y a trois concepts importants à maîtriser lorsqu'on utilise des commandes dans une base multi-process :
1. Chaque table est soit en mode "lecture seulement" soit en mode "lecture/écriture".
2. Les enregistrements sont verrouillés quand ils sont chargés et déverrouillés quand ils sont libérés.
3. Un enregistrement verrouillé ne peut être modifié.

Dans les paragraphes suivants, la personne effectuant une opération dans la base multi-utilisateurs est l'utilisateur local. Les autres personnes utilisant la base sont appelées les autres utilisateurs. La discussion est menée du point de vue de l'utilisateur local. De même, du point de vue du multi-process, le process exécutant une opération dans la base est le process courant. Tout autre process en cours d'exécution est désigné comme un autre process. La discussion est menée du point de vue du process courant.

Enregistrements verrouillés

Un enregistrement verrouillé ne peut pas être modifié par l'utilisateur local ou le process courant. Un enregistrement verrouillé peut être chargé, mais pas modifié. Un enregistrement est verrouillé quand l'un des autres utilisateurs ou process a chargé l'enregistrement pour effectuer une modification. Seul l'utilisateur qui modifie l'enregistrement perçoit l'enregistrement comme étant non verrouillé. Tous les autres utilisateurs et process perçoivent l'enregistrement comme étant verrouillé, et donc indisponible pour modification. Une table doit être en mode lecture/écriture pour qu'un enregistrement puisse être chargé déverrouillé (modifiable).

Etats Lecture seulement et Lecture/écriture

Chaque table d'une base est soit en mode lecture/écriture, soit en mode lecture seulement pour chaque utilisateur et process de la base. **Lecture seulement** signifie que les enregistrements de la table peuvent être chargés mais non modifiés. **Lecture/écriture** signifie que les enregistrements de la table peuvent être chargés et modifiés par un utilisateur/process, si aucun autre utilisateur/process n'a prélablement verrouillé l'enregistrement.

Notez que si vous changez l'état d'une table, celui-ci prend effet pour le prochain enregistrement chargé. Si un enregistrement est déjà chargé, la modification de l'état de la table ne l'affecte pas.

Etat lecture seulement
Lorsqu'une table est en lecture seulement et qu'un enregistrement est chargé, l'enregistrement est toujours verrouillé. En d'autres termes, l'enregistrement peut être affiché, imprimé et utilisé de diverses façons, mais ne peut être modifié.

Notez que le mode lecture seulement ne s'applique qu'à la modification d'enregistrements existants. Le mode lecture seulement n'affecte pas la création de nouveaux enregistrements : vous pouvez toujours ajouter des enregistrements à une table en lecture seulement en utilisant les commandes CREER ENREGISTREMENT et AJOUTER ENREGISTREMENT ou les commandes de menu du mode Développement.

4D met automatiquement une table en mode lecture seulement pour les commandes qui ne requièrent pas d'accès en écriture aux enregistrements. Ces commandes sont VISUALISER SELECTION, VALEURS DISTINCTES, ECRITURE DIF, ECRITURE SYLK, EXPORTER TEXTE, GRAPHE SUR SELECTION, IMPRIMER SELECTION, IMPRIMER ETIQUETTES, QR ETAT, SELECTION VERS TABLEAU, SELECTION LIMITEE VERS TABLEAU.

Vous pouvez connaître à tout moment l'état d'une table à l'aide de la fonction Etat lecture seulement.

Avant d'exécuter ces commandes, 4D sauvegarde l'état courant de la table (lecture seulement ou lecture/écriture) dans le process courant. Une fois la commande exécutée, l'état initial est rétabli.

Etat lecture/écriture
Lorsqu'une table est en lecture/écriture et qu'un enregistrement est chargé, l'enregistrement sera non verrouillé si aucun autre utilisateur ne l'a préalablement chargé. Si l'enregistrement est verrouillé par un autre utilisateur, l'enregistrement est chargé verrouillé et ne peut être modifié par l'utilisateur local.

Une table doit être en mode lecture/écriture et l'enregistrement doit être chargé pour qu'il soit déverrouillé et donc modifiable.

Si un utilisateur charge un enregistrement d'une table en mode lecture/écriture, aucun autre utilisateur ne peut charger cet enregistrement pour modification. Les autres utilisateurs peuvent toujours, cependant, ajouter des enregistrements dans la table, soit manuellement en mode Développement, soit par les commandes CREER ENREGISTREMENT ou AJOUTER ENREGISTREMENT.

Le mode lecture/écriture est le mode par défaut pour toutes les tables quand une base est ouverte et un nouveau process démarré.

Changer l'état d'une table

Vous pouvez utiliser les commandes LECTURE SEULEMENT et LECTURE ECRITURE pour changer l'état d'une table. Si vous voulez changer l'état d'une table pour mettre un enregistrement en lecture seulement ou lecture/écriture, vous devez exécuter la commande avant le chargement de l'enregistrement. Un enregistrement déjà chargé n'est pas affecté par les commandes LECTURE SEULEMENT et LECTURE ECRITURE.

Chaque process dispose de son propre état (lecture seulement ou lecture/écriture) pour chaque table dans la base.

Charger, modifier et libérer des enregistrements

Pour que l'utilisateur local puisse modifier un enregistrement, la table doit être en mode lecture/écriture et l'enregistrement doit être chargé et non verrouillé.

Chacune des commandes chargeant un enregistrement courant (s'il y en a un) — telles que ENREGISTREMENT SUIVANT, CHERCHER, TRIER, CHARGER SUR LIEN, etc — influe sur l'état de l'enregistrement. L'enregistrement est chargé en fonction de l'état courant de sa table (lecture seulement ou lecture/écriture) et sa propre disponibilité. Un enregistrement peut aussi être chargé d'une table liée par une commande activant le lien automatique.

Lorsqu'une table est en mode lecture seulement, tout enregistrement chargé de cette table est verrouillé. Un enregistrement verrouillé ne peut être sauvegardé ou supprimé depuis un autre process. Le mode lecture seulement est le mode recommandé puisqu'il autorise les autres utilisateurs à charger, modifier, et ensuite sauvegarder l'enregistrement.

Lorsqu'une table est en mode lecture/écriture, tout enregistrement chargé de cette table est déverrouillé seulement si aucun autre utilisateur ne l'a préalablement verrouillé. Un enregistrement déverrouillé peut être modifié et sauvegardé. Une table doit être placée en mode lecture/écriture avant qu'on ait besoin de charger, modifier et sauvegarder l'enregistrement.

Si un enregistrement doit être modifié, utilisez la fonction Enregistrement verrouille pour tester s'il est ou non verrouillé par un autre utilisateur. Si l'enregistrement est verrouillé (Enregistrement verrouille retourne Vrai), chargez l'enregistrement avec la commande CHARGER ENREGISTREMENT et testez de nouveau le verrouillage. Répétez l'opération jusqu'à ce que l'enregistrement soit libéré (Enregistrement verrouille retourne Faux).

Lorsque vous en avez terminé avec les modifications à apporter à un enregistrement, il doit être libéré pour les autres utilisateurs (et donc déverrouillé) avec la commande LIBERER ENREGISTREMENT. Si un enregistrement n'est pas libéré, il reste verrouillé pour tous les autres utilisateurs jusqu'à ce qu'un nouvel enregistrement courant soit sélectionné. Changer l'enregistrement courant d'une table libère automatiquement l'enregistrement courant précédent. Vous devez explicitement appeler LIBERER ENREGISTREMENT si vous ne changez pas l'enregistrement courant. Ce principe ne s'applique qu'aux enregistrements existants : quand un enregistrement est créé, il peut être sauvegardé quel que soit l'état de la table auquel il appartient.

Note : Lorsqu'elle est utilisée dans une transaction, la commande LIBERER ENREGISTREMENT libère l'enregistrement courant pour le process qui gère la transaction uniquement. Pour les autres process, l'enregistrement reste verrouillé tant que la transaction n'est pas validée (ou annulée).

Utilisez la commande VERROUILLE PAR pour savoir quel utilisateur ou process a verrouillé un enregistrement.

Boucles pour charger des enregistrements non verrouillés

Cet exemple présente la boucle la plus simple pour charger un enregistrement non verrouillé :

```
LECTURE ECRITURE ([Clients])  ` Placer la table en lecture/écriture
Repeter  ` Boucler jusqu'à ce que l'enregistrement soit déverrouillé
   CHARGER ENREGISTREMENT ([Clients])  ` Charger et verrouiller l'enregistrement
Jusque (Non (Enregistrement verrouille([Clients])))
```

La boucle s'exécute jusqu'à ce que l'enregistrement soit libéré.

Une boucle de ce type ne s'emploie que lorsqu'il est peu probable que l'enregistrement soit verrouillé par quelqu'un d'autre, puisque l'utilisateur aurait à attendre la fin de la boucle. Aussi, une telle boucle est rarement utilisée, sauf si l'enregistrement n'est modifiable que par méthode.

Cet exemple utilise la boucle précédente pour charger un enregistrement non verrouillé et modifier l'enregistrement :

```
LECTURE ECRITURE([Stocks])
Repeter  ` Boucle jusqu'à ce que l'enregistrement soit déverrouillé
    CHARGER ENREGISTREMENT([Stocks])  ` Charger l'enregistrement et le verrouiller
Jusque (Non (Enregistrement verrouille([Stocks])))
[Stocks]Part Qté := [Stocks]Part Qté – 1  ` Modifier l'enregistrement
STOCKER ENREGISTREMENT ([Stocks])  ` Sauvegarder l'enregistrement
LIBERER ENREGISTREMENT ([Stocks])  ` Laisser d'autres utilisateurs le modifier
LECTURE SEULEMENT([Stocks])
```

La commande MODIFIER ENREGISTREMENT prévient automatiquement l'utilisateur si un enregistrement est verrouillé, et interdit sa modification. L'exemple suivant évite la notification automatique par un test préalable de l'enregistrement avec la fonction Enregistrement verrouille. Si l'enregistrement est verrouillé, l'utilisateur peut annuler.

Cet exemple teste si l'enregistrement courant est verrouillé pour la table [Commandes]. Si c'est le cas, le process est endormi par la méthode pendant quelques instants. Cette technique peut être utlisée à la fois dans un développement multi-utilisateurs et multi-process :

```
Repeter
    LECTURE SEULEMENT([Commandes])
        ` Vous n'avez pas besoin de lecture/écriture pour le moment
    CHERCHER([Commandes])
        ` Si la recherche est terminée et que des enregistrements sont retournés
    Si ((OK=1) & (Enregistrements trouves([Commandes])>0))
        LECTURE ECRITURE([Commandes]) ` Mettre la table en mode lecture/écriture
        CHARGER ENREGISTREMENT([Commandes])
        Tant que (Enregistrement verrouille([Commandes]) & (OK=1))
                ` Si l'enregistrement est verrouillé,
                ` boucler jusqu'à ce que l'enregistrement soit libéré
                ` Par qui l'enregistrement est-il verrouillé ?
            VERROUILLE PAR([Commandes];$Process;$Utilisateur;$Machine;$Nom)
            Si ($Process=-1) ` L'enregistrement a-t-il été détruit?
                ALERTE("L'enregistrement a été détruit dans l'intervalle.")
                OK:=0
            Sinon
                Si ($Utilisateur="") ` Etes en mode simple utilisateur ?
                    $Utilisateur:="vous-même"
                Fin de si
```

```
                CONFIRMER("L'enregistrement est utilisé par "+$Utilisateur+" dans le "
                                                                    +$Nom+" Process.")
                Si (OK=1) ` Si vous voulez attendre quelques secondes
                    ENDORMIR PROCESS(Numero du process courant;120)  `Attente
                    CHARGER ENREGISTREMENT([Commandes])
                        ` Essayez de charger l'enregistrement
                Fin de si
            Fin de si
        Fin tant que
        Si (OK=1)   ` L'enregistrement est libéré
            MODIFIER ENREGISTREMENT([Commandes])
                ` Vous pouvez modifier l'enregistrement
            LIBERER ENREGISTREMENT([Commandes])
        Fin de si
        LECTURE SEULEMENT([Commandes]) ` Retour à lecture seulement
        OK:=1
    Fin de si
Jusque (OK=0)
```

Comportement des commandes en cas d'enregistrement verrouillé

Certaines commandes du langage effectuent des actions particulières lorsqu'elles rencontrent un enregistrement verrouillé. Voici la liste de ces commandes :

• MODIFIER ENREGISTREMENT : Cette commande affiche une boîte de dialogue indiquant que l'enregistrement est utilisé. L'enregistrement n'est pas affiché et donc l'utilisateur ne peut le modifier. En mode Développement, l'enregistrement est visible en lecture seulement.
• MODIFIER SELECTION : Cette commande se comporte normalement à ceci près que lorsque l'utilisateur double-clique sur un enregistrement pour le modifier, MODIFIER SELECTION affiche une boîte de dialogue indiquant que l'enregistrement est utilisé et n'autorise que sa lecture.
• APPLIQUER A SELECTION : Cette commande charge un enregistrement verrouillé, mais ne le modifie pas. APPLIQUER A SELECTION peut être utilisée pour lire les informations de la table sans problème. Si la commande rencontre un enregistrement verrouillé, celui-ci est placé dans l'ensemble système LockedSet.
• SUPPRIMER SELECTION : Cette commande ne supprime pas l'enregistrement verrouillé, elle l'ignore simplement. L'enregistrement verrouillé est placé dans l'ensemble LockedSet.
• SUPPRIMER ENREGISTREMENT : Cette commande est ignorée si l'enregistrement est verrouillé. Aucune erreur n'est retournée. Vous devez vous assurer que l'enregistrement est non verrouillé avant d'exécuter cette commande.

• STOCKER ENREGISTREMENT : Cette commande est ignorée si l'enregistrement est verrouillé. Aucune erreur n'est retournée. Vous devez vous assurer que l'enregistrement est non verrouillé avant d'exécuter cette commande.
• TABLEAU VERS SELECTION : Cette commande ne sauvegarde pas les enregistrements verrouillés. Si elle rencontre un enregistrement verrouillé, celui-ci est placé dans l'ensemble LockedSet.
• ALLER A ENREGISTREMENT : Dans une base multi-utilisateurs/multi-process, des enregistrements peuvent être ajoutés ou supprimés par d'autres utilisateurs/process. Les numéros d'enregistrement peuvent donc varier. Soyez prudent lorsque vous référencez un enregistrement par son numéro dans une base multi-utilisateurs.
• Ensembles : Soyez prudent lors de la manipulation d'ensembles puisque les informations sur lesquelles était basée la construction de l'ensemble peuvent avoir été modifiées par un autre utilisateur ou process. Pour plus d'informations sur les ensembles, reportez-vous à la section Présentation des ensembles.

Référence

CHARGER ENREGISTREMENT, Enregistrement verrouille, Etat lecture seulement, LECTURE ECRITURE, LECTURE SEULEMENT, LIBERER ENREGISTREMENT, Méthodes, Variables, VERROUILLE PAR.

CHARGER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle charger l'enregistrement courant ou Table par défaut si ce paramètre est omis

Description

CHARGER ENREGISTREMENT charge l'enregistrement courant de table. S'il n'y a pas d'enregistrement courant, CHARGER ENREGISTREMENT ne fait rien.

Il est alors utile d'appeler la fonction Enregistrement verrouille pour déterminer si l'enregistrement peut être modifié :
• Si la table est en mode Lecture seulement, la fonction retourne Vrai et vous ne pouvez pas modifier l'enregistrement.
• Si la table est en mode Lecture/écriture mais si l'enregistrement est déjà verrouillé, il sera en mode Lecture seulement et vous ne pouvez pas le modifier.
• Si la table est en mode Lecture/écriture et si l'enregistrement n'est pas verrouillé, vous pouvez le modifier dans le process courant. La fonction Enregistrement verrouille retournera Vrai pour tous les autres utilisateurs et process.

Note : Si la commande CHARGER ENREGISTREMENT est exécutée après un LECTURE SEULEMENT, l'enregistrement est automatiquement libéré et chargé, sans qu'il soit nécessaire d'appeler la commande LIBERER ENREGISTREMENT.

Vous n'aurez normalement pas besoin d'appeler la commande CHARGER ENREGISTREMENT, car toutes les commandes telles que CHERCHER, ENREGISTREMENT SUIVANT, ENREGISTREMENT PRECEDENT, etc., chargent automatiquement l'enregistrement courant.

En environnements multi-utilisateurs et multi-process, lorsque vous devez modifier un enregistrement existant, il vous faut accéder en Lecture/écriture à la table à laquelle appartient l'enregistrement. Lorsqu'un enregistrement verrouillé ne peut être chargé, CHARGER ENREGISTREMENT vous permet de tenter à nouveau plus tard de charger l'enregistrement. En utilisant CHARGER ENREGISTREMENT dans une boucle, vous pouvez attendre que l'enregistrement devienne accessible en Lecture/écriture.

Astuce : La commande CHARGER ENREGISTREMENT peut être utilisée pour recharger l'enregistrement courant dans le contexte d'un formulaire entrée. Toutes les données modifiées sont alors remplacées par leurs valeurs précédentes. Dans ce cas, la commande CHARGER ENREGISTREMENT effectue en quelque sorte une annulation globale de la saisie.

Référence

Enregistrement verrouille, LIBERER ENREGISTREMENT, Verrouillage d'enregistrements.

Enregistrement verrouille {(table)} → Booléen

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement dont vous voulez tester le verrouillage ou Table par défaut si ce paramètre est omis
Résultat	Booléen	←	L'enregistrement est verrouillé (Vrai) ou L'enregistrement n'est pas verrouillé (Faux)

Description

Enregistrement verrouille teste si l'enregistrement courant de table est verrouillé. Cette fonction vous permet de savoir si un enregistrement est verrouillé ou non, et donc de réagir de manière appropriée, par exemple en laissant à l'utilisateur le choix d'attendre que l'enregistrement soit libéré ou d'annuler l'opération.

Si Enregistrement verrouille retourne Vrai, l'enregistrement est verrouillé par un autre utilisateur ou un autre process et ne peut être sauvegardé. Dans ce cas, vous devez appeler la commande CHARGER ENREGISTREMENT pour tenter à nouveau de charger l'enregistrement, jusqu'à ce que Enregistrement verrouille retourne Faux.

Si Enregistrement verrouille retourne Faux, l'enregistrement n'est pas verrouillé, ce qui signifie qu'il est verrouillé pour tous les autres utilisateurs. Seul l'utilisateur ayant chargé l'enregistrement ou le process courant peut modifier et sauvegarder l'enregistrement. Une table doit être en mode lecture/écriture si vous voulez modifier les enregistrements qu'elle contient.

Lorsque vous tentez de charger un enregistrement qui a été supprimé, Enregistrement verrouille continue de retourner Vrai. Pour éviter d'attendre un enregistrement qui n'existe plus, appelez la commande VERROUILLE PAR. Cette commande retourne -1 dans le paramètre process si l'enregistrement a été supprimé.

Note : Enregistrement verrouille retourne Faux lorsqu'il n'y a pas d'enregistrement courant dans table, c'est-à-dire lorsque Numero enregistrement retourne -1.

Au cours d'une transaction, CHARGER ENREGISTREMENT et Enregistrement verrouille sont souvent appelées pour tester la disponibilité des enregistrements. Si un enregistrement est verrouillé, il suffit d'annuler la transaction.

Référence

CHARGER ENREGISTREMENT, Verrouillage d'enregistrements, VERROUILLE PAR.

Etat lecture seulement {(table)} → Booléen

Paramètre	Type		Description
table	Table	→	Table pour laquelle il faut tester l'état ou Table par défaut si ce paramètre est omis
Résultat	Booléen	←	Accès à la table est lecture seulement (Vrai) ou Accès à la table est lecture-écriture (Faux)

Description

La fonction Etat lecture seulement est utilisé pour tester si table est en mode lecture seulement dans le process où la fonction est appelée. Etat lecture seulement retourne Vrai si table est en lecture seulement, et Faux si table est en lecture-écriture.

Exemples

L'exemple suivant teste le statut de la table [Factures]. Si elle est en lecture seulement, le mode lecture-écriture lui est appliqué et l'enregistrement courant est rechargé.

```
Si (Etat lecture seulement([Factures]))
   LECTURE ECRITURE([Factures])
   CHARGER ENREGISTREMENT([Factures])
Fin de si
```

Note : L'enregistrement courant est rechargé pour permettre à l'utilisateur de le modifier. En effet, un enregistrement précédemment chargé en mode lecture seulement reste verrouillé jusqu'à ce qu'il soit rechargé en mode lecture-écriture.

Référence

LECTURE ECRITURE, LECTURE SEULEMENT, Verrouillage d'enregistrements.

LECTURE ECRITURE {(table | *)}

Paramètre	Type		Description
table | *	Table	→	Table à définir en mode lecture/écriture ou * pour toutes les tables ou Table par défaut si ce paramètre est omis

Description

LECTURE ECRITURE place table en mode lecture/écriture pour le process dans lequel la commande a été appelée. Si vous passez le paramètre optionnel *, toutes les tables sont placées en mode lecture/écriture.

Après un appel à LECTURE ECRITURE, lorsqu'un enregistrement est chargé, il n'est pas verrouillé — sauf si un autre utilisateur l'a déjà chargé. Cette commande ne modifie pas le statut des enregistrements déjà chargés, seuls les enregistrements chargés par la suite sont affectés.

Par défaut, toutes les tables sont en mode lecture/écriture.

Utilisez LECTURE ECRITURE lorsque vous devez modifier un enregistrement et sauvegarder les modifications. Vous pouvez également appeler cette commande lorsque vous voulez qu'un enregistrement soit verrouillé pour les autres utilisateurs, même si vous ne souhaitez pas effectuer de modifications. Placer une table en mode lecture/écriture vous permet d'empêcher les autres utilisateurs d'effectuer des modifications sur cette table. Cependant, ils peuvent continuer à créer des nouveaux enregistrements.

Note : Cette commande n'est pas rétroactive. Les privilèges de lecture/écriture pour un enregistrement sont définis par ceux de la table au moment où l'enregistrement est chargé. Pour qu'un enregistrement soit chargé en mode lecture/écriture alors que la table est en mode lecture seulement, vous devez placer la table en mode lecture/écriture avant que l'enregistrement soit chargé.

Référence

Etat lecture seulement, LECTURE SEULEMENT, Verrouillage d'enregistrements.

LECTURE SEULEMENT {(table | *)}

Paramètre	Type		Description
table | *	Table	→	Table à définir en mode lecture seulement ou * pour toutes les tables ou Table par défaut si ce paramètre est omis

Description

LECTURE SEULEMENT place table en mode lecture seulement pour le process dans lequel la commande a été appelée. Tous les enregistrements chargés par la suite sont verrouillés, aucune modification ne peut leur être apportée. Si vous passez le paramètre optionnel *, toutes les tables sont placées en mode lecture seulement.

Vous pouvez utiliser LECTURE SEULEMENT lorsqu'il n'est pas utile de modifier les enregistrements.

Note : Cette commande n'est pas rétroactive. Les privilèges de lecture/écriture pour un enregistrement sont définis par ceux de la table au moment où l'enregistrement est chargé. Pour qu'un enregistrement soit chargé en mode lecture seulement alors que la table est en mode lecture/écriture, vous devez placer la table en mode lecture seulement avant que l'enregistrement soit chargé.

Référence

Etat lecture seulement, LECTURE ECRITURE, Verrouillage d'enregistrements.

LIBERER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table pour laquelle l'enregistrement est à libérer ou Table par défaut si ce paramètre est omis

Description

LIBERER ENREGISTREMENT place l'enregistrement courant de table dans l'état non verrouillé.

Si l'enregistrement n'est pas verrouillé par un utilisateur (mais est verrouillé pour les utilisateurs), LIBERER ENREGISTREMENT libère l'enregistrement pour tous les utilisateurs.

Même si LIBERER ENREGISTREMENT libère l'enregistrement de la mémoire, celui-ci reste l'enregistrement courant. Lorsqu'un enregistrement devient l'enregistrement courant, le précédent est automatiquement libéré et n'est donc plus verrouillé pour les autres utilisateurs. Vous devez appeler cette commande lorsque vous avez fini de modifier un enregistrement, que vous voulez qu'il reste l'enregistrement courant et qu'il soit accessible aux autres utilisateurs.

Si les enregistrements contiennent une quantité importante de données, de champs Image ou de documents externes (tels que des documents 4D Write ou 4D Draw), il est préférable de ne pas stocker l'enregistrement courant en mémoire sauf si vous avez besoin de le modifier. Dans ce cas, il faut utiliser la commande LIBERER ENREGISTREMENT. De cette manière, vous pouvez conserver l'enregistrement courant sans qu'il soit en mémoire. Vous libérez ainsi la mémoire occupée par l'enregistrement, mais vous n'avez pas accès aux valeurs stockées dans les champs. Si vous avez besoin d'exploiter ces valeurs, il faut utiliser la commande CHARGER ENREGISTREMENT.

Note : Lorsqu'elle est utilisée dans une transaction, la commande LIBERER ENREGISTREMENT libère l'enregistrement courant pour le process qui gère la transaction uniquement. Pour les autres process, l'enregistrement reste verrouillé tant que la transaction n'est pas validée (ou annulée).

Référence

CHARGER ENREGISTREMENT, Verrouillage d'enregistrements.

VERROUILLE PAR ({table; }process; utilisateur4D; utilisateurSession; nomProcess)

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement verrouillé ou Table par défaut si ce paramètre est omis
process	Numérique	←	Numéro du process sur le serveur
utilisateur4D	Alpha	←	Nom de l'utilisateur 4D
utilisateurSession	Alpha	←	Nom de l'utilisateur ayant ouvert la session de travail
nomProcess	Alpha	←	Nom du process

Description

VERROUILLE PAR retourne des informations sur l'utilisateur et le process qui ont verrouillé l'enregistrement. Le numéro du process (sur le poste serveur), le nom de l'utilisateur dans l'application 4D et dans le système ainsi que le nom du process sont retournés dans les variables process, utilisateur4D, utilisateurSession et nomProcess. Vous pouvez utiliser ces informations dans une boîte de dialogue pour avertir l'utilisateur lorsqu'un enregistrement est verrouillé.

Si l'enregistrement n'est pas verrouillé, process prend la valeur 0 et utilisateur4D, utilisateurSession et nomProcess retournent des chaînes vides. Si vous essayez de charger en lecture/écriture un enregistrement qui a été supprimé, process retourne -1 et utilisateur4D, utilisateurSession et nomProcess retournent des chaînes vides.

Le paramètre utilisateur4D est le nom de l'utilisateur défini dans l'éditeur de mots de passe de 4D. Si aucun mot de passe n'a été défini, "Super_Utilisateur" est retourné.

Le paramètre utilisateurSession retourné correspond au nom de l'utilisateur ayant ouvert la session sur le poste client (ce nom est notamment affiché dans la fenêtre d'administration de 4D Server pour chaque process ouvert).

Référence

Enregistrement verrouille, Verrouillage d'enregistrements.

15

Ensembles

Les ensembles sont un moyen puissant et rapide de manipuler des sélections d'enregistrements. En plus de la possibilité de créer des ensembles, de les relier à la sélection courante, de les stocker, de les charger et de les effacer, 4D permet d'effectuer trois opérations standard sur les ensembles :
• Intersection,
• Union,
• Différence.

Ensembles et sélection courante

Un ensemble est une représentation d'une sélection d'enregistrements. L'idée d'ensemble est intimement liée à celle de sélection courante. Les ensembles sont généralement utilisés pour les raisons suivantes :
• Sauvegarder et ensuite restaurer une sélection lorsque l'ordre n'a pas d'importance,
• Accéder à la sélection que l'utilisateur a faite à l'écran (l'ensemble UserSet),
• Réaliser une opération logique entre des sélections.

La sélection courante est une liste de références qui pointent vers chaque enregistrement actuellement sélectionné. La liste existe en mémoire. Seuls les enregistrements sélectionnés sont dans la liste. Une sélection ne contient pas en réalité les enregistrements, mais seulement une liste de références à ces enregistrements. Chaque référence à un enregistrement utilise 4 octets en mémoire. Lorsque vous travaillez sur une table, vous travaillez toujours avec les enregistrements de la sélection courante. Lorsqu'une sélection est triée, seule la liste des références est réorganisée. Il n'y a qu'une sélection courante pour chaque table dans un process.

Comme une sélection courante, un ensemble représente une sélection d'enregistrements. Un ensemble le fait en utilisant une représentation très compacte pour chaque enregistrement. En effet, chacun est représenté par un bit (un huitième d'octet). Les opérations utilisant les ensembles sont très rapides parce les ordinateurs accomplissent très rapidement les opérations sur les bits. Un ensemble contient un bit pour chaque enregistrement de la table, que l'enregistrement soit inclus dans l'ensemble ou non. En fait, chaque bit est égal à 1 ou 0 — tout dépend si l'enregistrement est dans l'ensemble ou non.

Les ensembles sont très économiques en termes de mémoire RAM. La taille d'un ensemble, en octets, est toujours égale au nombre total des enregistrements dans la table divisé par huit. Ainsi, pour une table contenant 10 000 enregistrements, l'ensemble prend 1250 octets, ce qui fait environ 1,2 Ko de RAM.

Il peut exister plusieurs ensembles pour chaque table. En fait, les ensembles peuvent être sauvegardés sur disque indépendamment de la base. Pour modifier un enregistrement appartenant à un ensemble, vous devez d'abord utiliser l'ensemble comme sélection courante, puis modifier l'enregistrement.

Un ensemble n'est jamais trié — les enregistrements sont simplement marqués comme appartenant ou non à l'ensemble. En revanche, une sélection temporaire est triée, mais requiert davantage de mémoire dans la plupart des cas. Pour plus d'information sur les sélections temporaires, reportez-vous à la section Sélections temporaires.

Au moment de sa création, un ensemble “mémorise” l'enregistrement courant de la sélection.

Comparaison	Sélection courante	Ensembles
Nombre par table	1	illimité
Triable	Oui	Non
Sauvegardable sur disque	Non	Oui
RAM par enreg. (en octets)	Nb enreg. sélec*4	Nb total enreg./8
Combinable	Non	Oui
Contient enreg. courant	Oui	Oui, celui du moment de création

L'ensemble que vous créez appartient à la table dans laquelle il a été créé. Les opérations sur les ensembles ne peuvent être effectuées qu'entre ensembles appartenant à la même table.

Les ensembles sont indépendants des données, ce qui signifie qu'après des modifications dans une table, un ensemble peut n'être plus exact. Bien des opérations peuvent rendre un ensemble inexact. Si vous créez un ensemble de tous les habitants de New York et changez ensuite les données de l'un des enregistrements par “New Jersey,” l'ensemble ne change pas puisqu'il est simplement la représentation d'une sélection d'enregistrements. L'ajout ou la suppression d'enregistrements peut également rendre un ensemble obsolète, de même que le compactage des données. Les ensembles ne sont exacts que tant que la sélection d'origine n'est pas modifiée.

Ensembles process et interprocess

Vous pouvez utiliser trois types d'ensembles :

• **Ensembles process** : Un ensemble process n'est accessible que par le process dans lequel il a été créé. L'ensemble système LockedSet est un ensemble process. Les ensembles process sont effacés dès que la méthode du process est achevée. Les ensembles process ne requièrent pas de préfixe pour leur nom.

• **Ensembles interprocess** : Un ensemble est interprocess lorsque son nom est précédé des symboles (<>) — le signe “inférieur à” suivi du signe “supérieur à”. Cette syntaxe est utilisable à la fois sous Windows et Mac OS. Sous Mac OS, vous pouvez aussi utiliser le symbole “diamant” (Option+v sur un clavier français).
Un ensemble interprocess est “visible” par tous les process de la base.
En client/serveur, un ensemble interprocess est “visible” par tous les process de tous les clients s'il a été créé dans un process client ou par tous les process serveur s'il a été créé dans un process serveur (procédure stockée).
Le nom d'un ensemble interprocess doit être unique dans la base.

• **Ensembles locaux/Ensembles client** : Les ensembles locaux/client sont principalement destinés au mode client/serveur. Leur nom est toujours précédé du symbole dollar ($) — à l'exception de l'ensemble système UserSet. A la différence des autres types d'ensembles, un ensemble local/client est stocké sur le poste client.

Note : Pour plus d'informations sur l'utilisation des ensembles en client/serveur, reportez-vous à la section 4D Server, ensembles et sélections dans le *Guide de référence* de 4D Server.

Visibilité des ensembles
Le tableau suivant indique les principes de visibilité des ensembles en fonction de leur portée et de leur lieu de création :

	Process client	Autres process du client	Autres clients	Process serveur	Autres process du serveur
Création dans un process client					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Création dans un process serveur					
$test				x	
test				x	
<>test				x	x

Ensembles et transactions

Un ensemble peut être créé à l'intérieur d'une transaction. Il est donc possible de définir un "ensemble des enregistrements créés pendant la transaction" et un "ensemble des enregistrements créés ou modifiés en-dehors de la transaction". Une fois la transaction terminée, l'ensemble créé pendant la transaction doit être effacé car il pourrait ne plus être une représentation exacte des enregistrements, surtout si la transaction a été annulée.

Exemple d'ensemble

Cet exemple détruit des enregistrements doublons dans une table. La table contient des informations sur des personnes. Une structure répétitive Boucle...Fin de boucle parcourt tous les enregistrements, comparant l'enregistrement courant à l'enregistrement précédent. Si le nom, l'adresse et le code postal sont identiques, l'enregistrement est ajouté à un ensemble. A la fin de la boucle, l'ensemble devient la sélection courante et l'ancienne sélection courante est détruite :

```
ENSEMBLE VIDE([Personnes];"Doublons")
    ` Créer un ensemble vide pour les doublons
TOUT SELECTIONNER([Personnes])
    ` Tous les enregistrements
    ` Trier les enregistrements par code postal, adresse et nom pour
    ` que les doublons soient les uns à côté des autres
TRIER ([Personnes];[Personnes]CODEPOSTAL;>;[Personnes]Adresse;>;[Personnes]Nom;>)
    ` Initialiser les variables conservant les champs des enregistrements précédents
$Nom:=[Personnes]Nom
$Adresse:=[Personnes]Adresse
$CODEPOSTAL:=[Personnes]CODEPOSTAL
    ` Aller au second enregistrement pour le comparer au premier
ENREGISTREMENT SUIVANT ([Personnes])
Boucle ($i; 2; Enregistrements dans table ([Personnes]))
        ` Parcourir les enregistrements en partant à 2
        ` Si nom, adresse et CODEPOSTAL sont identiques au
        ` précédent enregistrement, alors c'est un doublon.
    Si (([Personnes]Nom=$Nom) & ([Personnes]Adresse=$Adresse) &
                                        ([Personnes]CODEPOSTAL=$CODEPOSTAL))
            ` Ajouter enregistrement (le doublon) à l'ensemble
        ADJOINDRE ELEMENT ([Personnes]; "Doublon")
    Sinon
            ` Garder les nom, adresse et CODEPOSTAL de cet enregistrement pour
            ` comparer avec le prochain
        $Nom:=[Personnes]Nom
        $Adresse:=[Personnes]Adresse
        $CODEPOSTAL:=[Personnes]CODEPOSTAL
    Fin de si
        ` Passer à l'enregistrement suivant
    ENREGISTREMENT SUIVANT([Personnes])
Fin de boucle
```

```
    ` Utiliser doublons trouvés (en tant que sélection courante)
UTILISER ENSEMBLE ("Doublons")
    ` Détruire doublons
SUPPRIMER SELECTION ([Personnes])
    ` Supprimer l'ensemble de la mémoire
EFFACER ENSEMBLE ("Doublons")
```

Au lieu de supprimer immédiatement les enregistrements à la fin de la méthode, vous pouvez les afficher à l'écran ou les imprimer si des comparaisons plus fines doivent être menées.

Ensemble système UserSet

4D gère un ensemble système local/client nommé UserSet. UserSet contient automatiquement l'ensemble des derniers enregistrements marqués (“surlignés”) à l'écran par l'utilisateur dans un formulaire en liste. Ainsi, vous pouvez afficher un groupe d'enregistrements avec MODIFIER SELECTION ou VISUALISER SELECTION, demander à l'utilisateur d'en sélectionner certains et retourner le résultat de cette sélection manuelle dans un ensemble que vous nommez ou dans une sélection.

Il existe un seul UserSet par process. Les tables ne disposent pas de leur propre UserSet. Le UserSet n'est associé à une table que lorsqu'une sélection d'enregistrements est affichée pour cette table.

4D gère l'ensemble UserSet pour les formulaires liste affichés en mode Développement et via les commandes MODIFIER SELECTION ou VISUALISER SELECTION. En revanche, ce mécanisme n'est pas actif pour les sous-formulaires.

La méthode ci-dessous illustre comment afficher des enregistrements pour permettre à l'utilisateur d'en sélectionner quelques-uns, et ensuite utiliser le UserSet pour afficher les enregistrements sélectionnés :

```
    ` Afficher tous les enregistrements et permettre à l'utilisateur d'en sélectionner un
    ` certain nombre. Puis afficher cette sélection en utilisant UserSet pour modifier la
    ` sélection courante.
FORMULAIRE SORTIE([Personnes]; "Display") ` Choisir le formulaire sortie
TOUT SELECTIONNER ([Personnes]) ` Sélection de toutes les personnes
ALERTE ("Appuyer Ctrl ou Commande + Clic pour sélectionner des enregistrements.")
VISUALISER SELECTION ([Personnes]) ` Afficher les personnes
UTILISER ENSEMBLE ("UserSet") ` Utiliser les personnes sélectionnées
ALERTE ("Vous avez choisi les personnes suivantes.")
VISUALISER SELECTION ([Personnes]) ` Afficher les personnes sélectionnées
```

4D Server : Bien que son nom ne débute pas par le caractère "$", l'ensemble système UserSet est un ensemble client. Par conséquent, lors de l'utilisation des commandes INTERSECTION, REUNION et DIFFERENCE, veillez à ne comparer UserSet qu'à d'autres ensembles clients. Pour plus d'informations, reportez-vous aux descriptions de ces commandes ainsi qu'à la section 4D Server, ensembles et sélections dans le *Guide de référence* de 4D Server.

Ensemble système LockedSet

Les commandes APPLIQUER A SELECTION, TABLEAU VERS SELECTION et SUPPRIMER SELECTION créent un ensemble système nommé LockedSet lorsqu'elles sont utilisées en environnement multiprocess.
Les commandes de recherche créent également un ensemble systèmeLockedSet lorsqu'elles trouvent des enregistrements verrouillés dans le contexte de 'recherche et verrouillage' (cf. commande FIXER RECHERCHE ET VERROUILLAGE).
LockedSet indique quels enregistrements étaient verrouillés lors de l'exécution d'une commande.

Référence

Nommer les objets du langage 4D.

ADJOINDRE ELEMENT

version 5

ADJOINDRE ELEMENT ({table; }ensemble)

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement courant ou Table par défaut si ce paramètre est omis
ensemble	Alpha	→	Nom de l'ensemble auquel ajouter l'enregistrement courant

Description

ADJOINDRE ELEMENT ajoute l'enregistrement courant de table à ensemble. L'ensemble doit avoir déjà été créé ; si ensemble n'existe pas, une erreur est retournée. S'il n'y a pas d'enregistrement courant pour table, ADJOINDRE ELEMENT ne fait rien.

Référence

ENLEVER ELEMENT.

Appartient a ensemble

Appartient a ensemble (ensemble) → Booléen

Paramètre	Type		Description
ensemble	Alpha	→	Nom de l'ensemble à tester
Résultat	Booléen	←	L'enregistrement courant est dans l'ensemble (Vrai) ou l'enregistrement courant n'est pas dans l'ensemble (Faux)

Description

Appartient a ensemble teste si l'enregistrement courant de la table est inclus dans ensemble. La fonction Appartient a ensemble retourne Vrai si l'enregistrement courant de la table est dans ensemble, et retourne Faux si l'enregistrement courant de la table n'est pas dans ensemble.

Exemple

L'exemple suivant est la méthode objet d'un bouton testant si l'enregistrement courant est inclus dans l'ensemble des meilleurs clients :

```
Si (Appartient a ensemble ("Meilleurs"))
    ALERTE ("C'est un de nos meilleurs clients.")
Sinon
    ALERTE ("Ce n'est pas un de nos meilleurs clients.")
Fin de si
```

Référence

ADJOINDRE ELEMENT, ENLEVER ELEMENT.

CHARGER ENSEMBLE

CHARGER ENSEMBLE ({table; }ensemble; document)

Paramètre	Type		Description
table	Table	→	Table à laquelle appartient l'ensemble ou Table par défaut si ce paramètre est omis
ensemble	Alpha	→	Nom de l'ensemble à créer en mémoire
document	Alpha	→	Document disque contenant l'ensemble

Description

CHARGER ENSEMBLE charge un ensemble depuis le fichier document, créé à l'aide de la commande STOCKER ENSEMBLE.

L'ensemble stocké dans document doit s'appliquer à table. Si ensemble existait déjà en mémoire, il est réécrit.

Le paramètre document est le nom du fichier disque contenant l'ensemble. Il n'est pas nécessaire que ce fichier ait le même nom que l'ensemble. Si vous passez une chaîne vide dans document, une boîte de dialogue standard d'ouverture de fichiers s'affiche, permettant à l'utilisateur de choisir l'ensemble à charger.

ATTENTION : Rappelez-vous qu'un ensemble est l'image d'une sélection d'enregistrements au moment précis où l'ensemble est créé. Si les enregistrements représentés par l'ensemble sont modifiés, celui-ci devient obsolète. En conséquence, vous devez stocker et charger des ensembles avec des enregistrements dont le contenu varie peu. De multiples événements peuvent rendre un ensemble obsolète : modification ou suppression d'un enregistrement de l'ensemble, ou encore modification des critères ayant déterminé la création de l'ensemble.

Exemple

L'exemple suivant utilise CHARGER ENSEMBLE pour charger l'ensemble des locaux de l'entreprise Dupont SARL à Paris :

```
    ` Charger l'ensemble en mémoire
CHARGER ENSEMBLE ([Entreprises]; "Paris Dupont SARL"; "PaDupontEns")
    ` Modifier la sélection courante avec l'ensemble
UTILISER ENSEMBLE ("Paris Dupont SARL")
EFFACER ENSEMBLE ("Paris Dupont SARL") ` Effacer l'ensemble de la mémoire
```

Référence

STOCKER ENSEMBLE.

Variables et ensembles système

Si l'utilisateur clique sur Annuler dans la boîte de dialogue d'ouverture de fichiers, ou si une erreur se produit pendant le chargement, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

COPIER ENSEMBLE

COPIER ENSEMBLE (srcEns; dstEns)

Paramètre	Type		Description
srcEns	Alpha	→	Nom de l'ensemble source
dstEns	Alpha	→	Nom de l'ensemble de destination

Description

La commande COPIER ENSEMBLE copie le contenu de l'ensemble srcEns dans l'ensemble dstEns.

Les deux ensembles peuvent être process, interprocess ou locaux.

Exemples

(1) L'exemple suivant, en client/serveur, copie l'ensemble local "$SetA", conservé sur le poste client, vers l'ensemble process "SetB", conservé sur le poste serveur :

```
COPIER ENSEMBLE("$SetA";"SetB")
```

(2) L'exemple suivant, en client/serveur, copie l'ensemble process "SetA", conservé sur le poste serveur, vers l'ensemble local "$SetB", conservé sur le poste client :

```
COPIER ENSEMBLE("SetA";"$SetB")
```

Référence

Présentation des ensembles.

CREER ENSEMBLE SUR TABLEAU (table; tabEnrg{; nomEns})

Paramètre	Type		Description
table	Table	→	Table de l'ensemble
tabEnrg	Tab Entier long | Booléen	→	Tableau de n° d'enregistrements, ou Tableau de booléens (Vrai = l'enregistrement est dans l'ensemble, Faux = il n'est pas dans l'ensemble)
nomEns	Alpha	→	Nom de l'ensemble à créer, ou Appliquer la commande à l'ensemble Userset si ce paramètre est omis ou vide

Description

La commande CREER ENSEMBLE SUR TABLEAU crée l'ensemble nomEns à partir :
• soit du tableau de numéros d'enregistrements absolus tabEnrg de la table table,
• soit du tableau de booléens tabEnrg ; dans ce cas, les valeurs du tableau indiquent l'appartenance (VRAI) ou non (FAUX) de chaque enregistrement de table à l'ensemble nomEns.

Lorsque vous utilisez la commande avec un tableau d'entiers longs, tous les numéros du tableau représentent la liste des numéros d'enregistrements qui feront partie de l'ensemble nomEns. Si un numéro est invalide (enregistrement non créé), l'erreur -10503 est générée.

Lorsque vous utilisez la commande avec un tableau de booléens, le Nième élément du tableau représente l'intégration (VRAI) ou non (FAUX) de l'enregistrement numéro N à l'ensemble nomEns. En principe, le nombre d'éléments du tableau doit être égal au nombre d'enregistrements de table. Si le tableau est plus petit que le nombre d'enregistrements, seuls les enregistrements définis par le tableau pourront faire partie de l'ensemble.

Note : Avec un tableau de booléens, la commande utilise les éléments à partir du numéro 0 jusqu'au numéro N-1.

Si vous ne passez pas le paramètre nomEns ou si vous passez une chaîne vide, la commande s'applique à l'ensemble système Userset.

Référence

CREER SELECTION SUR TABLEAU, Nommer les objets du langage 4D, TABLEAU BOOLEEN SUR ENSEMBLE.

Gestion des erreurs

Dans un tableau d'entier longs, si un numéro d'enregistrement est invalide (enregistrement non créé), l'erreur -10503 est générée.

DIFFERENCE (ensemble1; ensemble2; résultat)

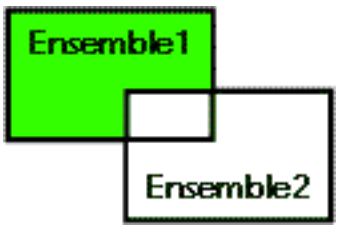
Paramètre	Type		Description
ensemble1	Alpha	→	Ensemble initial
ensemble2	Alpha	→	Ensemble à exclure
résultat	Alpha	→	Ensemble résultant

Description

DIFFERENCE fusionne ensemble1 et ensemble2 et exclut de l'ensemble résultat tous les enregistrements se trouvant dans ensemble2. Autrement dit, un enregistrement est inclus dans l'ensemble résultat s'il appartient à ensemble1 mais n'appartient pas à ensemble2. Le tableau suivant liste les résultats possibles d'une opération de différence d'ensembles.

Ensemble1	Ensemble2	Ensemble résultant
Oui	Non	Oui
Oui	Oui	Non
Non	Oui	Non
Non	Non	Non

Le schéma ci-dessous est la représentation graphique d'une opération de différence entre deux ensembles. La zone colorée est l'ensemble résultant.



L'ensemble résultat est créé par DIFFERENCE. Il remplace tout ensemble du même nom existant déjà, y compris ensemble1 et ensemble2. Les ensembles ensemble1 et ensemble2 doivent appartenir à la même table. L'ensemble résultat appartient à la même table que ensemble1 et ensemble2.

4D Server : En mode client/serveur, les ensembles sont "visibles" en fonction de leur type (interprocess, process et local) et de leur lieu de création (serveur ou client). DIFFERENCE requiert que les trois ensembles soient visibles sur la même machine. Pour plus d'informations sur ce point, reportez-vous au paragraphe 4D Server, ensembles et sélections dans le manuel de référence de 4D Server.

Exemple

L'exemple suivant exclut les enregistrements sélectionnés par l'utilisateur. Les enregistrements sont affichés à l'écran par l'instruction suivante :

```
VISUALISER SELECTION ([Clients]) ` Affichage des clients sous forme de liste
```

Un bouton associé à une méthode objet est placé en bas de la liste. La méthode objet exclut les enregistrements sélectionnés par l'utilisateur (l'ensemble système nommé UserSet) et affiche une sélection réduite :

```
NOMMER ENSEMBLE ([Clients]; "$Courant")
    ` Création d'un ensemble à partir de la sélection courante
DIFFERENCE ("$Courant";"UserSet";"$Courant")
    ` Exclusion des enregistrements sélectionnés
UTILISER ENSEMBLE ("$Courant") ` Utilisation du nouvel ensemble
EFFACER ENSEMBLE ("$Courant") ` Effacement de l'ensemble
```

Référence

INTERSECTION, REUNION.

EFFACER ENSEMBLE

EFFACER ENSEMBLE (ensemble)

Paramètre	Type		Description
ensemble	Alpha	→	Nom de l'ensemble à effacer de la mémoire

Description

EFFACER ENSEMBLE efface ensemble de la mémoire et la libère ainsi pour d'autres utilisations. EFFACER ENSEMBLE n'a aucune conséquence sur les tables, sélections ou enregistrements. Pour sauvegarder un ensemble avant de l'effacer, utiliser la commande STOCKER ENSEMBLE. Comme les ensembles consomment de la mémoire, pensez à les effacer dès qu'ils ne sont plus nécessaires.

Exemples

Reportez-vous à l'exemple de la commande UTILISER ENSEMBLE.

Référence

CHARGER ENSEMBLE, ENSEMBLE VIDE, NOMMER ENSEMBLE.

ENLEVER ELEMENT ({table; }ensemble)

Paramètre	Type		Description
table	Table	→	Table de l'enregistrement courant ou Table par défaut si ce paramètre est omis
ensemble	Alpha	→	Nom de l'ensemble duquel supprimer l'enregistrement courant

Description

ENLEVER ELEMENT supprime l'enregistrement courant de table de l'ensemble ensemble. L'ensemble doit déjà exister ; s'il n'existe pas, une erreur est générée. S'il n'y a pas d'enregistrement courant dans table, ENLEVER ELEMENT ne fait rien.

Référence

ADJOINDRE ELEMENT.

Enregistrements dans ensemble

Enregistrements dans ensemble (ensemble) → Numérique

Paramètre	Type		Description
ensemble	Alpha	→	Nom de l'ensemble à tester
Résultat	Numérique	←	Nombre d'enregistrements dans l'ensemble

Description

Enregistrements dans ensemble retourne le nombre d'enregistrements présents dans ensemble. Si ensemble n'existe pas ou s'il n'y a pas d'enregistrements dans ensemble, Enregistrements dans ensemble retourne 0.

Exemple

L'exemple suivant affiche une boîte de dialogue d'alerte qui indique le pourcentage des clients qui sont considérés comme les meillleurs :

```
    ` Calculer d'abord le pourcentage
$Pourcent := (Enregistrements dans ensemble ("Meilleurs") / Enregistrements dans table
                                                    ([Clients])) * 100
    ` Afficher une alerte avec le pourcentage
ALERTE (Chaine ($Pourcent; "##0%") + " de nos clients sont nos meilleurs clients.")
```

Référence

Enregistrements dans table, Enregistrements trouves.

ENSEMBLE VIDE ({table; }ensemble)

Paramètre	Type		Description
table	Table	→	Table pour laquelle créer un ensemble vide ou Table par défaut si ce paramètre est omis
ensemble	Alpha	→	Nom du nouvel ensemble vide

Description

ENSEMBLE VIDE crée un ensemble vide, ensemble, pour table. Vous pouvez ajouter des enregistrements dans cet ensemble à l'aide de la commande ADJOINDRE ELEMENT. Si un ensemble du même nom existe déjà, il est effacé et remplacé par le nouvel ensemble.

Note : Il n'est pas indispensable d'appeler la commande ENSEMBLE VIDE avant d'utiliser la commande NOMMER ENSEMBLE.

Exemple

Reportez-vous à l'exemple proposé dans la section Présentation des ensembles.

Référence

EFFACER ENSEMBLE, NOMMER ENSEMBLE.

INTERSECTION (ensemble1; ensemble2; résultat)

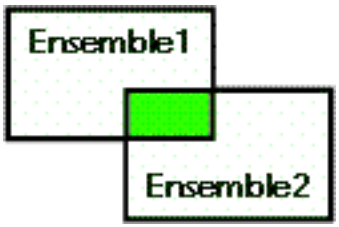
Paramètre	Type		Description
ensemble1	Alpha	→	Premier ensemble
ensemble2	Alpha	→	Second ensemble
résultat	Alpha	→	Ensemble résultant

Description

INTERSECTION compare ensemble1 et ensemble2 et sélectionne uniquement les enregistrements se trouvant à la fois dans ensemble1 et dans ensemble2. Le tableau suivant liste les résultats possibles d'une opération d'intersection d'ensembles.

Ensemble1	Ensemble2	Ensemble résultant
Oui	Non	Non
Oui	Oui	Oui
Non	Oui	Non
Non	Non	Non

Le schéma ci-dessous est la représentation graphique de l'intersection de deux ensembles. La zone colorée est l'ensemble résultant.



L'ensemble résultat est créé par INTERSECTION. Il remplace tout ensemble du même nom existant déjà, y compris ensemble1 et ensemble2. Les ensembles de départ ensemble1 et ensemble2 doivent appartenir à la même table. L'ensemble résultat appartient à la même table que ensemble1 et ensemble2.

4D Server : En mode client/serveur, les ensembles sont "visibles" en fonction de leur type (interprocess, process et local) et de leur lieu de création (serveur ou client). INTERSECTION requiert que les trois ensembles soient visibles sur la même machine. Pour plus d'informations sur ce point, reportez-vous au paragraphe 4D Server, ensembles et sélections dans le manuel de référence de 4D Server.

Exemple

L'exemple suivant recherche les clients en contact avec deux représentants, Jean et Grégoire. Chaque représentant dispose d'un ensemble regroupant ses clients. Les clients se trouvant dans les deux ensembles sont en contact avec Jean et Grégoire :

```
INTERSECTION ("Jean"; "Grégoire"; "Doublon")
    ` Doublon reçoit les clients appartenant aux 2 ensembles
UTILISER ENSEMBLE ("Doublon") ` Modification de la sélection courante
EFFACER ENSEMBLE ("Doublon") ` Effacement de cet ensemble mais sauvegarde des autres
VISUALISER SELECTION ([Clients])
    ` Affichage des clients en contact avec les deux commerciaux
```

Référence

DIFFERENCE, REUNION.

NOMMER ENSEMBLE

NOMMER ENSEMBLE ({table; }ensemble)

Paramètre	Type		Description
table	Table	→	Table pour laquelle vous voulez créer un ensemble à partir de la sélection courante ou Table par défaut si ce paramètre est omis
ensemble	Alpha	→	Nom du nouvel ensemble

Description

NOMMER ENSEMBLE crée un nouvel ensemble, ensemble, pour table, et y place la sélection courante. Le pointeur d'enregistrement courant de la table est sauvegardé avec ensemble. Si ensemble est passé à la commande UTILISER ENSEMBLE, la sélection courante et l'enregistrement courant sont restitués. Comme pour tout ensemble, il ne peut y avoir de tri, et lorsque ensemble est appelé, l'ordre par défaut est utilisé. Si un ensemble du même nom existe déjà, il est effacé et remplacé par le nouvel ensemble.

Exemples

L'exemple suivant crée un ensemble après qu'une recherche ait été effectuée, de manière à ce que l'ensemble puisse être stocké sur disque :

```
CHERCHER ([Personnes])  ` L'utilisateur effectue une recherche
    ` Création d'un nouvel ensemble
NOMMER ENSEMBLE ([Personnes]; "EnsembleRecherche")
    ` L'ensemble est stocké sur disque
STOCKER ENSEMBLE ("EnsembleRecherche"; "MaRecherche")
```

Référence

EFFACER ENSEMBLE, ENSEMBLE VIDE, Nommer les objets du langage 4D.

REUNION (ensemble1; ensemble2; résultat)

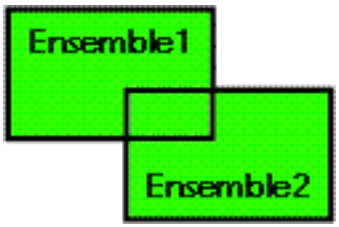
Paramètre	Type		Description
ensemble1	Alpha	→	Premier ensemble
ensemble2	Alpha	→	Second ensemble
résultat	Alpha	→	Ensemble résultant

Description

REUNION crée un nouvel ensemble contenant tous les enregistrements de ensemble1 et ensemble2. Le tableau suivant liste les résultats possibles d'une opération de réunion d'ensembles.

Ensemble1	Ensemble2	Ensemble résultant
Oui	Non	Oui
Oui	Oui	Oui
Non	Oui	Oui
Non	Non	Non

Le schéma ci-dessous est la représentation graphique de la réunion de deux ensembles. La zone colorée est l'ensemble résultant.



L'ensemble résultat est créé par REUNION. Il remplace tout ensemble du même nom existant déjà, y compris ensemble1 et ensemble2. Les ensembles de départ ensemble1 et ensemble2 doivent appartenir à la même table. L'ensemble résultat appartient à la même table que ensemble1 et ensemble2. L'enregistrement courant de résultat est celui de ensemble1.

4D Server : En mode client/serveur, les ensembles sont "visibles" en fonction de leur type (interprocess, process et local) et de leur lieu de création (serveur ou client). REUNION requiert que les trois ensembles soient visibles sur la même machine. Pour plus d'informations sur ce point, reportez-vous au paragraphe 4D Server, ensembles et sélections dans le manuel de référence de 4D Server.

Exemple

L'exemple suivant ajoute des enregistrements à l'ensemble des meilleurs clients. Les enregistrements sont affichés à l'écran. Ensuite, l'ensemble des meilleurs clients est chargé du disque, et tous les enregistrements sélectionnés par l'utilisateur (l'ensemble système UserSet) sont ajoutés. Enfin, le nouvel ensemble est sauvegardé sur le disque :

```
TOUT SELECTIONNER ([Clients]) ` Sélection de tous les enregistrements
VISUALISER SELECTION ([Clients]) ` Afficher tous les clients en mode liste
CHARGER ENSEMBLE ("$Meilleurs"; "$Meilleurs.sav")
    ` Chargement de l'ensemble des meilleurs clients
REUNION ("$Meilleurs"; "UserSet"; "$Meilleurs") ` Ajout de toute sélection à l'ensemble
STOCKER ENSEMBLE ("$Meilleurs"; "$Meilleurs.sav")
    ` Sauvegarde de l'ensemble des meilleurs clients
```

Référence

DIFFERENCE, INTERSECTION.

STOCKER ENSEMBLE (ensemble; document)

Paramètre	Type		Description
ensemble	Alpha	→	Nom de l'ensemble à stocker
document	Alpha	→	Nom du fichier dans lequel stocker l'ensemble

Description

STOCKER ENSEMBLE sauvegarde ensemble dans le fichier disque document.

Il n'est pas nécessaire que document ait le même nom que l'ensemble. Si vous passez une chaîne vide dans document, une boîte de dialogue standard de sauvegarde de fichiers apparaît, permettant à l'utilisateur de saisir un nom de fichier. Vous pourrez utiliser la commande CHARGER ENSEMBLE pour charger un ensemble stocké sur disque.

Si l'utilisateur clique sur le bouton Annuler dans la boîte de dialogue de sauvegarde de fichiers, ou si une erreur se produit lors de la sauvegarde, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

La commande STOCKER ENSEMBLE est souvent utilisée pour stocker sur disque les résultats d'une recherche particulièrement longue.

ATTENTION : Rappelez-vous qu'un ensemble est l'image d'une sélection d'enregistrements au moment précis où l'ensemble est créé. Si les enregistrements représentés par l'ensemble sont modifiés, celui-ci devient obsolète. En conséquence, vous devez créer et sauvegarder des ensembles représentant des enregistrements dont le contenu varie peu. De multiples événements peuvent rendre un ensemble obsolète : modification ou suppression d'un enregistrement de l'ensemble, ou encore modification des critères ayant déterminé la création de l'ensemble. Rappelez-vous également que les ensembles ne stockent pas les valeurs des champs.

Exemple

L'exemple suivant affiche la boîte de dialogue standard d'enregistrement de fichiers afin de permettre à l'utilisateur de saisir le nom du fichier contenant l'ensemble :

```
STOCKER ENSEMBLE ("UnEnsemble"; "")
```

Référence

CHARGER ENSEMBLE.

Variables et ensembles système

Si l'utilisateur clique sur le bouton Annuler dans la boîte de dialogue standard de sauvegarde de documents, ou si une erreur se produit pendant la sauvegarde, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

UTILISER ENSEMBLE (ensemble)

Paramètre	Type		Description
ensemble	Alpha	→	Nom de l'ensemble à utiliser

Description

UTILISER ENSEMBLE crée, avec les enregistrements de ensemble, une nouvelle sélection courante pour la table à laquelle ensemble appartient.

Au moment où vous créez un ensemble, la position de l'enregistrement courant est sauvegardée. UTILISER ENSEMBLE récupère cette information et fait de l'enregistrement le nouvel enregistrement courant. Si vous supprimez cet enregistrement avant d'exécuter UTILISER ENSEMBLE, 4D sélectionne comme enregistrement courant le premier enregistrement de l'ensemble. Les commandes du thème "Ensembles", INTERSECTION, REUNION, DIFFERENCE, et ADJOINDRE ELEMENT réinitialisent l'enregistrement courant.
Si vous avez créé un ensemble ne contenant pas de position d'enregistrement courant, UTILISER ENSEMBLE désigne le premier enregistrement de l'ensemble comme enregistrement courant.

ATTENTION : Rappelez-vous qu'un ensemble est la représentation d'une sélection d'enregistrements à un instant donné (au moment de la création de l'ensemble). Si les enregistrements que l'ensemble représente sont modifiés, il se peut que celui-ci ne soit plus valide. En conséquence, un ensemble sauvegardé sur disque doit généralement représenter un groupe d'enregistrements qui ne change pas souvent. De multiples événements peuvent rendre un ensemble invalide, comme par exemple la suppression ou la modification d'un enregistrement de l'ensemble, ou encore la modification des critères de création de l'ensemble.

Exemple

L'exemple suivant utilise CHARGER ENSEMBLE pour charger un ensemble des sites de la société Dubois à Paris. UTILISER ENSEMBLE est ensuite appelée pour faire de l'ensemble la sélection courante :

```
` Charger l'ensemble en mémoire
CHARGER ENSEMBLE ([Entreprises]; "DuboisParis"; "ENSDuboisParis")
UTILISER ENSEMBLE ("DuboisParis") ` Modification de la sélection courante
EFFACER ENSEMBLE ("DuboisParis") ` Effacement de l'ensemble de la mémoire
```

Référence

CHARGER ENSEMBLE, EFFACER ENSEMBLE.

16

Environnement 4D

AJOUTER SEGMENT DE DONNEES

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Note de compatibilité : Depuis la version 11 de 4D, les segments de données ne sont plus pris en charge (la taille du fichier de données est désormais illimitée). Lorsqu'elle est appelée, cette commande ne fait rien.

Compacter fichier donnees (cheminStructure; cheminDonnées{; dossierArchive{; options{; méthode}}}) → Texte

Paramètre	Type		Description
cheminStructure	Texte	→	Chemin d'accès du fichier de structure
cheminDonnées	Texte	→	Chemin d'accès du fichier de données
dossierArchive	Texte	→	Chemin d'accès du dossier dans lequel placer le fichier de données original
options	Num	→	Options de compactage
méthode	Texte	→	Nom de la méthode 4D de rétro-appel
Résultat	Texte	←	Chemin d'accès complet du dossier contenant le fichier de données original

Description

La commande Compacter fichier donnees effectue un compactage du fichier de données désigné par le paramètre cheminDonnées associé au fichier de structure cheminStructure. Pour plus d'informations sur le compactage, reportez-vous au manuel *Mode Développement*.

Pour assurer la continuité du fonctionnement de la base, le nouveau fichier de données compacté remplace automatiquement le fichier original. Par sécurité, le fichier original n'est pas modifié et est déplacé dans un dossier spécial nommé "*Replaced files (compacting) AAAA-MM-JJ HH-MM-SS*" où *AAAA-MM-JJ HH-MM-SS* représente la date et l'heure de la sauvegarde. Par exemple : "Replaced files (compacting) 2007-09-27 15-20-35".

La commande retourne le chemin d'accès complet du dossier effectivement créé pour stocker le fichier de données original. Cette commande peut être exécutée depuis 4D (mode local) ou 4D Server uniquement (procédure stockée). Le fichier de données à compacter doit correspondre au fichier de structure désigné par cheminStructure. En outre, il ne doit PAS être ouvert au moment de l'exécution de la commande, sinon une erreur est générée.
Si une erreur se produit durant le processus de compactage, les fichiers originaux sont conservés à leur emplacement initial. Si un fichier d'index (fichier .4DIndx) est associé au fichier de données, il est également compacté. Comme pour le fichier de données, le fichier original est sauvegardé et la nouvelle version compactée remplace la précédente.

• Passez dans cheminStructure le chemin d'accès complet du fichier de structure associé au fichier de données que vous souhaitez compacter. Cette information est nécessaire à la procédure de compactage. Le chemin d'accès doit être exprimé dans la syntaxe du système d'exploitation. Vous pouvez également passer une chaîne vide, dans ce cas une boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de désigner le fichier de structure à utiliser.

• Vous pouvez passer dans cheminDonnées une chaîne vide, un nom de fichier ou un chemin d'accès complet, exprimé dans la syntaxe du système d'exploitation. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichier apparaît, permettant à l'utilisateur de désigner le fichier de données à compacter. Ce fichier doit correspondre au fichier de structure défini dans le paramètre cheminStructure. Si vous passez uniquement un nom de fichier de données, 4D le recherchera au même niveau que le fichier de structure.

• Le paramètre facultatif dossierArchive permet de désigner l'emplacement du dossier "Replaced files (compacting) *Dateheure*" destiné à recueillir les versions originales des fichiers de données ainsi que des éventuels fichiers d'index.
La commande retourne le chemin d'accès complet du dossier effectivement créé.
- Si vous omettez ce paramètre, les fichiers d'origine sont automatiquement déplacés dans un dossier "Replaced files (compacting) *Dateheure*" créé à côté du fichier de structure.
- Si vous passez une chaîne vide, une boîte de dialogue standard d'ouverture de dossier apparaît, permettant à l'utilisateur de désigner l'emplacement du dossier à créer.
- Si vous passez un chemin d'accès (exprimé dans la syntaxe du système d'exploitation), la commande créera le dossier "Replaced files (compacting) *Dateheure*" à cet emplacement.

• Le paramètre facultatif options permet de définir diverses options liées au compactage. Pour cela, utilisez les constantes suivantes, placées dans le thème "Maintenance fichier de données". Vous pouvez passer les deux options en les cumulant :
- Ne pas créer d'historique (16384) : En principe, la commande Compacter fichier donnees crée un fichier d'historique au format xml (reportez-vous à la fin de la description de la commande). Vous pouvez annuler ce fonctionnement en passant cette option.
- Créer process (32768) : Lorsque cette option est passée, le compactage sera asynchrone et vous devrez gérer les résultats à l'aide de la méthode de rétro-appel (voir ci-dessous). 4D n'affichera pas de barre de progression (il est possible de le faire via la méthode de rétro-appel). La variable système *OK* prendra la valeur 1 si le process a été correctement lancé et 0 dans les autres cas. Lorsque cette option n'est pas passée, la variable *OK* prendra la valeur 1 si le compactage s'est correctement déroulé et sinon 0.

• Le paramètre méthode permet de désigner une méthode de rétro-appel qui sera régulièrement appelée durant le compactage si l'option Créer process a été passée. Dans le cas contraire, la méthode de rétro-appel n'est jamais appelée. Pour plus d'informations sur cette méthode, reportez-vous à la description de la commande VERIFIER FICHIER DONNEES. Si la méthode de rétro-appel n'existe pas dans la base, une erreur est générée et la variable système *OK* prend la valeur 0.

Par défaut, la commande Compacter fichier donnees crée un fichier d'historique au format xml (si vous n'avez pas passé l'option Ne pas créer d'historique, cf. paramètre options). Son nom est basé sur celui du fichier de données et il est placé à côté de ce fichier. Par exemple, pour un fichier de données nommé "data.4dd", le fichier d'historique sera nommé "Struct_Data_Compact_Log.xml".

Exemple

L'exemple suivant (Windows) effectue le compactage d'un fichier de données :

```
$ficStruc:=Fichier structure
$ficDonnées:="C:\Bases\Factures\Janvier\Factures.4dd"
$ficOrig:="C:\Bases\Factures\Archives\Janvier\"
$dossierArch:=Compacter fichier donnees($ficStruc;$ficDonnées;$ficOrig)
```

Référence

VERIFIER FICHIER DONNEES.

Variables et ensembles système

Si l'opération de compactage s'est déroulée correctement, la variable système *OK* prend la valeur 1, sinon elle prend la valeur 0.

CREER FICHIER DONNEES (cheminAccès)

Paramètre	Type		Description
cheminAccès	Alpha	→	Nom ou chemin d'accès complet du fichier de données à créer

Description

La commande CREER FICHIER DONNEES permet de créer un nouveau fichier de données sur disque et de remplacer à la volée le fichier de données ouvert par l'application 4D.

Le fonctionnement général de cette commande est identique à celui de la commande OUVRIR FICHIER DONNEES, à la différence près que le nouveau fichier de données désigné par le paramètre cheminAccès est créé juste après la réouverture du fichier de structure.

Avant de lancer l'opération, la commande vérifie que le chemin spécifié ne correspond pas à un fichier existant.

4D Server : Cette commande ne peut pas être utilisée avec 4D Client ou 4D Server.

Référence

OUVRIR FICHIER DONNEES.

Dossier 4D ({dossier}{; }{*}) → Alpha

Paramètre	Type		Description
dossier	Entier long	→	Type de dossier (si omis=dossier 4D actif)
*	*	→	Retourner le dossier de la base hôte
Résultat	Alpha	←	Chemin d'accès du dossier désigné

Description

La commande Dossier 4D renvoie le chemin d'accès du dossier 4D actif de l'application courante, ou du dossier de l'environnement 4D spécifié par le paramètre dossier, s'il est passé. Cette commande vous permet d'obtenir avec certitude le chemin d'accès réel des dossiers utilisés par l'application. En utilisant cette commande, vous êtes certain que votre code fonctionnera correctement sur toute plate-forme, quelles que soient la langue du système et l'application 4D.

Vous pouvez passer dans dossier une des constantes suivantes, placées dans le thème “Environnement 4D” :

Constante	Type	Valeur
Dossier 4D actif	Entier long	0 (défaut)
Dossier Licenses	Entier long	1
Dossier Extras	Entier long	2
Dossier base 4D Client	Entier long	3
Dossier base	Entier long	4
Dossier base syntaxe Unix	Entier long	5
Dossier Resources courant	Entier long	6
Dossier Logs	Entier long	7
Dossier racine HTML	Entier long	8

Chaque dossier est décrit ci-dessous.

Notes préalables sur les noms de dossiers :
• {Disque} est le disque sur lequel est installé le système.
• Le libellé Utilisateur représente le nom de l'utilisateur ayant ouvert la session.
• Avec certaines versions de Mac OS, les noms des dossiers sont traduits :
- le dossier Library est nommé Bibliothèque,
- le dossier Application Support est nommé Support aux applications.

Dossier 4D actif
Les applications de l'environnement 4D utilisent le dossier **4D** pour stocker les informations suivantes :
• Fichiers de préférences utilisés par les applications 4D
• Fichier shortcuts.xml (raccourcis clavier personnalisés)
• Dossier Macros v2 (macros commandes de l'éditeur de méthodes)
• Dossier Favorites v11 (chemins d'accès des bases locales et distantes ayant été ouvertes)

Le dossier 4D actif se trouve par défaut à l'emplacement suivant :
• Sous Windows Vista : {Disque}:\Users*Utilisateur*\AppData\Roaming\4D
• Sous Windows XP : {Disque}:\Documents and Settings*Utilisateur*\Application Data\4D
• Sous Mac OS : {Disque}:Users:*Utilisateur*:Library:Preferences:4D

Dossier Licenses
Dossier contenant les fichiers de licence 4D du poste.
Le dossier **Licenses** est situé à l'emplacement suivant :
• Sous Windows Vista : {Disque}:\ProgramData\4D\Licenses
• Sous Windows XP : {Disque}:\Documents and Settings\All Users\Application Data\4D\Licenses
• Sous Mac OS : {Disque}:Library:Application Support:4D:Licenses

Notes :
• Dans le cas d'une application fusionnée avec un 4D Volume Desktop, le dossier des licences est inclus dans le *package* (progiciel) de l'applicatif.
• Si le dossier des licences n'a pu être créé dans le système à cause d'un défaut d'autorisation, il est créé aux emplacements suivants :
- sous Windows Vista : {Disque}:\Users*Utilisateur*\AppData\Roaming\4D\Licenses
- sous Windows XP : {Disque}:\Documents and Settings*Utilisateur*\Application Data\4D\Licenses
- sous Mac OS : {Disque}:Users:*Utilisateur*:Library:Application Support:4D:Licenses

Dossier Extras (obsolète)
Dossier au contenu personnalisé téléchargé sur chaque poste client.

Note de compatibilité : A compter de la version 11.2 de 4D v11 SQL, il est déconseillé d'utiliser le dossier Extras pour la communication personnalisée entre le serveur et les postes distants. Il est désormais recommandé d'utiliser pour cela le dossier Resources (cf. description du dossier Resources courant ci-dessous). Le dossier Extras reste toutefois pris en charge par 4D Server afin de préserver la compatibilité des applications existantes.

Note : Si le dossier Extras n'existe pas pour la base, l'exécution de la commande Dossier 4D avec la constante Dossier Extras provoque sa création.

Dossier base 4D Client (postes clients)
Dossier de la base 4D créé en local sur chaque poste client, dans lequel sont téléchargés depuis 4D Server les dossiers et fichiers relatifs à la base (ressources, plug-ins, dossier Resources, etc.). Le dossier **base 4D Client** est situé à l'emplacement suivant sur chaque poste client :
• Sous Windows Vista : {Disque}:\Users*Utilisateur*\AppData\Local\4D\NomDeLaBase_Adresse
• Sous Windows XP : {Disque}:\Documents and Settings*Utilisateur*\Local Settings\Application Data\NomDeLaBase_Adresse
• Sous Mac OS : {Disque}:Users:*Utilisateur*:Library:Caches:4D:NomDeLaBase_Adresse

Dossier base
Dossier contenant le fichier de structure de la base. Le chemin d'accès est exprimé avec la syntaxe standard de la plate-forme courante.
Avec l'application 4D Client, cette constante équivaut strictement à la constante précédente Dossier base 4D Client : la commande retourne le chemin d'accès du dossier créé en local.

Dossier base syntaxe Unix
Dossier contenant le fichier de structure de la base. Cette constante désigne le même dossier que la précédente, mais le chemin d'accès retourné est exprimé avec la syntaxe Unix (Posix), du type /Users/... Cette syntaxe est principalement utile lorsque vous utilisez la commande LANCER PROCESS EXTERNE sous Mac OS ou la commande FIXER EXECUTABLE CGI.

Dossier Resources courant
Dossier **Resources** de la base. Ce dossier contient les éléments additionnels (images, textes) utilisés pour l'interface de la base. Un composant peut disposer de son propre dossier Resources. Le dossier Resources est situé à côté du fichier de structure de la base.
En mode client/serveur, ce dossier permet d'organiser le transfert de données personnalisées (images, fichiers, sous-dossiers...) entre le poste serveur et les postes clients. Le contenu de ce dossier est mis à jour automatiquement sur chaque client au moment de sa connexion. Tous les mécanismes de référencement associé au dossier Resources sont pris en charge en mode client/serveur (dossier .lproj, XLIFF, images...) . En outre, 4D v11 SQL fournit divers outils permettant de gérer et de mettre à jour dynamiquement ce dossier, notamment un Explorateur de ressources.

Note : Si le dossier Resources n'existe pas pour la base, l'exécution de la commande Dossier 4D avec la constante Dossier Resources courant provoque sa création.

Dossier Logs
Dossier **Logs** de la base. Ce dossier centralise les fichiers d'historique de la base courante. Il est créé au même niveau que le fichier de structure. Le dossier Logs contient les fichiers d'historique suivants :
• conversion de la base,
• requêtes du serveur Web,
• vérification et réparation des données,

• vérification et réparation de la structure,
• journal d'activités sauvegarde/restitution,
• débogage des commandes,
• requêtes 4D Server (généré sur les clients et sur le serveur)..

Note : Si le dossier Logs n'existe pas pour la base, l'exécution de la commande Dossier 4D avec la constante Dossier Logs provoque sa création.

Dossier racine HTML
Dossier racine HTML courant de la base. Le chemin d'accès retourné est exprimé avec la syntaxe standard de la plate-forme courante. Le dossier racine HTML est le dossier dans lequel le serveur Web de 4D va chercher les pages et fichiers Web demandés. Par défaut, il est nommé *DossierWeb* et est placé à côté de fichier de structure (ou de sa copie locale avec 4D en mode distant). Son emplacement peut être défini dans la page Web/Configuration des Préférences ou dynamiquement via la commande FIXER RACINE HTML.
Si la commande Dossier 4D est appelée depuis un 4D distant, le chemin retourné est celui du poste distant, pas celui de 4D Server.

Le paramètre facultatif * est utile dans le cadre d'une architecture utilisant des composants : il permet de déterminer la base (hôte ou composant) dont vous souhaitez obtenir le chemin d'accès d'un dossier. Ce paramètre est valide uniquement pour les dossiers Dossier base, Dossier base syntaxe Unix et Dossier Resources courant. Il est ignoré dans les autres cas.
Lorsque la commande est appelée depuis un composant :
• si le paramètre * est passé, la commande retourne le chemin d'accès du dossier de la base hôte,
• si le paramètre * n'est pas passé, la commande retourne le chemin d'accès du dossier du composant.
Le dossier de la base (Dossier base et Dossier base syntaxe Unix) retourné diffère en fonction du type d'architecture du composant :
- dans le cas d'un dossier/package .4dbase, la commande retourne le chemin d'accès du dossier/package .4dbase,
- dans le cas d'un fichier .4db ou .4dc, la commande retourne le chemin d'accès du dossier "Components",
- dans le cas d'un alias ou raccourci, la commande retourne le chemin d'accès du dossier contenant la base matrice originale. Le résultat diffère en fonction du format de cette base (dossier/package .4dbase ou fichier .4db/.4dc), comme décrit ci-dessus.
Lorsque la commande est appelée depuis la base hôte, elle retourne toujours le chemin d'accès du dossier de la base hôte, que le paramètre * soit passé ou non.

Exemples

(1) Pendant le démarrage d'une base mono-utilisateur, vous voulez charger (ou créer) vos propres paramètres et les stocker dans un fichier situé dans le dossier 4D. Pour cela, dans la Méthode base Sur ouverture, vous pouvez écrire les lignes suivantes :

```
ASSOCIER TYPES FICHIER("PREF";"PRF";"Préférences")
    ` Associer le type de fichier PREF sur Mac OS à l'extension de fichier .PRF sur Windows
$vsNomDocPref:=Dossier 4D+"MesPrefs"`Construire le chemin d'accès au fichier Préférences
Si(Tester chemin acces($vsNomDocPref+(".PRF"*Num(Sous Windows)))#Est un document)
        ` Vérifier si le fichier existe
    $vtRefDocPref:=Creer document($vsNomDocPref;"PREF") ` Si non, il faut le créer
Sinon
    $vtRefDocPref:=Ouvrir document($vsNomDocPref;"PREF") ` Si oui, il faut l'ouvrir
Fin de si
Si (OK=1)   ` Traiter le contenu du document
    FERMER DOCUMENT($vtRefDocPref)
Sinon   ` Gérer l'erreur
Fin de si
```

(2) Cet exemple illustre l'emploi de la constante Dossier base syntaxe Unix sous Mac OS pour lister le contenu du dossier de la base :

```
$cheminposix:="\""+Dossier 4D(Dossier base syntaxe Unix)+"\""
$mondossier:="ls -l "+$cheminposix
$in:=""
$out:=""
$err:=""
LANCER PROCESS EXTERNE($mondossier;$in;$out;$err)
```

Note : Sous Mac OS, il est nécessaire d'encadrer les chemins d'accès avec des guillemets lorsqu'ils contiennent des noms de fichiers ou de dossiers comportant des espaces. La séquence d'échappement "\" permet d'insérer le caractère guillemets dans la chaîne. Vous pouvez également utiliser l'instruction Caractere(Guillemets).

Référence

Dossier systeme, Dossier temporaire, FIXER RACINE HTML, LISTE COMPOSANTS, Tester chemin acces.

Variables et ensembles système

Si le paramètre dossier est invalide ou si le chemin d'accès retourné est vide, la variable système *OK* prend la valeur 0.

ECRIRE CACHE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande ECRIRE CACHE sauvegarde immédiatement les caches de données sur le disque. Toutes les modifications apportées à la base sont alors stockées sur disque.

Généralement, vous n'avez pas besoin d'appeler cette commande, car 4D sauvegarde régulièrement les modifications. Il est préférable d'utiliser l'option **Ecriture cache toutes les X mn**(page Gestion des données des Préférences), qui spécifie les intervalles de sauvegarde des données, afin de contrôler l'écriture du cache de données sur le disque.

Note : 4D utilise en interne un système intégré de cache de données permettant d'accélérer les opérations d'E/S. Le fait que des modifications de données soient, par moments, présentes dans le cache de données et pas sur le disque est entièrement transparent pour votre code. Par exemple, si vous appelez la commande CHERCHER, le moteur de 4D va intégrer les données présentes dans le cache pour effectuer l'opération.

Fichier application → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom long du fichier 4D exécutable ou de l'application 4D

Description

La fonction Fichier application retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier exécutable ou de l'application 4D que vous utilisez.

Sous Windows
Si, par exemple, vous utilisez 4D qui se trouve dans le répertoire \PROGRAMMES\4D sur le volume E, Fichier application renvoie E:\PROGRAMMES\4D\4D.EXE.

Sous Mac OS
Si, par exemple, vous utilisez 4D qui se trouve dans le dossier Programmes sur le disque Disque Dur, Fichier application renvoie Disque Dur:Programmes:4D.app.

Exemple

Lorsque vous démarrez votre base sous Windows, vous souhaitez vérifier qu'une librairie DLL se trouve au même niveau que le fichier exécutable de 4D. Dans la Méthode base Sur ouverture, vous pouvez écrire les instructions suivantes :

```
Si (Sous Windows & (Type application#4D Server))
   Si (Tester chemin acces (Nom long vers chemin d'accès (Fichier application)+
                                                   "XRAYCAPT.DLL")#Est un document)
      ` Afficher une boîte de dialogue expliquant que la librairie XRAYCAPT.DLL n'est pas
      ` présente. Donc, la saisie de radios n'est pas disponible
   Fin de si
Fin de si
```

Note : Les méthodes projet *Sous Windows* et *Nom long vers chemin d'accès* sont détaillées dans la section Présentation des documents système.

Référence

Fichier donnees, Fichier structure, LISTE SEGMENTS DE DONNEES.

Fichier donnees

Fichier donnees {(segment)} → Alpha

Paramètre	Type		Description
segment	Entier long	→	Obsolète, ne pas utiliser
Résultat	Alpha	←	Nom long du fichier de données de la base

Description

La fonction Fichier donnees retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier de données de la base avec laquelle vous êtes en train de travailler.

Depuis la version 11 de 4D, les segments de données ne sont plus pris en charge. Le paramètre segment est désormais ignoré, il ne doit plus être utilisé.

Sous Windows
Si, par exemple, vous travaillez avec la base MesCDs qui se trouve à l'emplacement \DOCS\MesCDs sur le volume G, Fichier donnees retournera G:\DOCS\MesCDs\MesCDs.4DD (si vous avez choisi l'emplacement proposé par défaut par 4D lorsque vous avez créé la base).

Sous Mac OS
Si, par exemple, vous travaillez avec la base MesCDs qui se trouve dans le dossier Documents:MesCDs*f*: sur le disque Macintosh HD, Fichier donnees retournera Macintosh HD:Documents:MesCDs*f*:MesCDs.data (si vous avez choisi l'emplacement proposé par défaut par 4D lorsque vous avez créé la base).

ATTENTION : Si vous appelez cette fonction depuis 4D en mode distant, seul le nom du fichier de données est retourné, pas le nom long.

Référence

Fichier application, Fichier structure, LISTE SEGMENTS DE DONNEES.

Fichier donnees verrouille → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai = fichier/segment verrouillé Faux = fichier/segment non verrouillé

Description

La commande Fichier donnees verrouille retourne Vrai si le fichier de données de la base ouverte ou l'un de ses segments au moins est verrouillé — c'est-à-dire, protégé en écriture.

Placée par exemple dans la Méthode base Sur ouverture, cette commande permet de prévenir tout risque d'ouverture fortuite d'un fichier de données verrouillé.

Exemple

Cette méthode empêchera l'ouverture de la base si le fichier de données est verrouillé :

```
Si(Fichier donnees verrouille)
   ALERTE("Le fichier de données est verrouillé. Impossible d'ouvrir la base.")
   QUITTER 4D
Fin de si
```

Fichier structure {(*)} → Alpha

Paramètre	Type		Description
*	*	→	Retourner le fichier de structure de la base hôte
Résultat	Alpha	←	Nom long du fichier de structure de la base

Description

La fonction Fichier structure retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier de structure de la base en cours d'utilisation.

Sous Windows
Si, par exemple, vous travaillez sur la base MesCDs qui se trouve à \DOCS\MesCDs sur le volume G, Fichier structure renvoie G:\DOCS\MyCDs\MesCDs.4DB.

Sous Macintosh
Si, par exemple, vous travaillez sur la base MesCDs qui se trouve dans le dossier Documents:MesCDs*f*: sur le disque qui s'appelle Macintosh HD, Fichier structure renvoie Macintosh HD:Documents:MesCDs*f*:MesCDs.

Note : Dans le cas particulier d'une base compilée et fusionnée avec 4D Desktop, cette commande retourne le chemin d'accès du fichier de l'application (fichier exécutable) sous Windows et Mac OS. Sous Mac OS, ce fichier est situé à l'intérieur du progiciel, dans le dossier [Contents:MacOS]. Ce fonctionnement provient d'un ancien mécanisme, conservé pour des raisons de compatibilité. Si vous souhaitez obtenir le nom long du progiciel lui-même, il est préférable d'utiliser la commande Fichier application. L'astuce consiste à tester l'application à l'aide de la commande Type application puis à exécuter Fichier structure ou Fichier application en fonction du contexte.

ATTENTION : Si vous appelez cette commande lorsque vous utilisez 4D Client, seul le nom du fichier de structure est renvoyé, pas le nom long.

Le paramètre facultatif * est utile dans le cadre d'une architecture utilisant des composants : il permet de déterminer la structure (hôte ou composant) dont vous souhaitez obtenir le nom long en fonction du contexte d'appel de la commande :

• Lorsque la commande est appelée depuis un composant :
- si le paramètre * est passé, la commande retourne le nom long du fichier de structure de la base hôte,
- si le paramètre * n'est pas passé, la commande retourne le nom long du fichier de structure du composant.
Le fichier de structure d'un composant correspond au fichier .4db ou .4dc du composant situé dans le dossier "Components" de la base. Cependant, un composant peut également être installé sous la forme d'un alias/raccourci ou d'un dossier/package .4dbase :
- dans le cas d'un composant installé sous forme d'alias/raccourci, la commande retourne le chemin d'accès du fichier .4db ou .4dc original (l'alias ou le raccourci est résolu).
- dans le cas d'un composant installé sous forme de dossier/package .4dbase, la commande retourne le chemin d'accès du fichier .4db ou .4dc à l'intérieur de ce dossier/package.
• Lorsque la commande est appelée depuis une méthode d'une base hôte, elle retourne toujours le nom long du fichier de structure de la base hôte, que le paramètre * soit passé ou non.

Exemples

(1) Cet exemple affiche le nom et l'emplacement du fichier de structure que vous utilisez :

```
Si(Type application#4D Client)
    $vsStructNomFichier:=Nom long vers fichier(Fichier structure)
    $vsStructNomChemin:=Nom long vers chemin d'accès (Fichier structure)
    ALERTE("Vous êtes en train d'utiliser la base "+Caractere(34)+$vsStructNomFichier+
                Caractere(34)+" qui se trouve au "+Caractere(34)+$vsStructNomChemin+
                                                    Caractere(34)+".")
Sinon
    ALERTE("Vous êtes connecté à la base "+Caractere(34)+Fichier structure+
                                                    Caractere(34))
Fin de si
```

Note : Les méthodes projet Nom long vers fichier et Nom long vers chemin d'accès sont détaillées dans la section Présentation des documents système.

(2) L'exemple suivant permet de savoir si la méthode est appelée depuis un composant :

```
C_BOOLEEN($0)
$0:=(Fichier structure#Fichier structure(*))
    ` $0=Vrai si la méthode est appelée depuis un composant
```

Référence

Fichier application, Fichier donnees, LISTE COMPOSANTS, LISTE SEGMENTS DE DONNEES.

FIXER PARAMETRE BASE ({table; }sélecteur; valeur)

Paramètre	Type		Description
table	Table	→	Table à paramétrer ou Table par défaut si ce paramètre est omis
sélecteur	Entier long	→	Code du paramètre de la base à modifier
valeur	Entier long | Alpha	→	Valeur du paramètre

Description

La commande FIXER PARAMETRE BASE permet de modifier divers paramètres internes de la base de données 4D.

sélecteur désigne le paramètre à modifier. 4D vous propose des constantes prédéfinies, placées dans le thème “Paramètres de la base”. Le tableau suivant liste chaque constante, décrit sa portée et indique si les éventuelles modifications effectuées sont conservées entre deux sessions :

Constante sélecteur	Valeur	Portée	Conservation
Ratio de tri séq	1	**** *Sélecteur inactivé* ****	
Optimisation accès séq	2	**** *Sélecteur inactivé* ****	
Ratio valeurs distinctes séq	3	**** *Sélecteur inactivé* ****	
Compression index	4	**** *Sélecteur inactivé* ****	
Ratio chercher dans sélec séq	5	**** *Sélecteur inactivé* ****	
Minimum process Web	6	4D local, 4D Server (*)	Oui
Maximum process Web	7	4D local, 4D Server (*)	Oui
Mode conversion Web	8	Process courant	-
Taille cache données	9	Application 4D (*) (**)	-
Appels système 4D mode local	10	Application 4D (*)	Oui
Appels système 4D Server	11	Application 4D (*)	Oui
Appels système 4D mode distant	12	Application 4D (*)	Oui
Timeout 4D Server	13	Application 4D sauf si valeur négative (***)	Oui
Timeout 4D mode distant	14	Application 4D sauf si valeur négative (***)	Oui
Numéro du port	15	4D local, 4D Server (*)	-
Adresse IP d’écoute	16	4D local, 4D Server (*)	Oui
Jeu de caractères	17	4D local, 4D Server (*)	Oui
Process Web simultanés maxi	18	4D local, 4D Server (*)	Oui
Client Minimum process Web	19	Tous postes 4D distants (*)	Oui
Client Maximum process Web	20	Tous postes 4D distants (*)	Oui
Client Taille max requêtes Web	21	Tous postes 4D distants (*)	Oui
Client Numéro de port	22	Tous postes 4D distants (*)	Oui

Client Adresse IP d'écoute	23	Tous postes 4D distants (*)	Oui
Client Jeu de caractères	24	Tous postes 4D distants (*)	Oui
Client Proc Web simultanés maxi	25	Tous postes 4D distants (*)	Oui
Mode écriture cache	26	**** *Sélecteur inactivé* ****	
Taille maximum requêtes Web	27	4D local, 4D Server (*)	Oui
Enreg requêtes 4D Server	28	4D Server, 4D distant (*)	-
Enreg requêtes Web	29	4D local, 4D Server (*)	Oui
Client Enreg requêtes Web	30	Tous postes 4D distants (*)	Oui
Numéro automatique table	31	Application 4D	Oui
Précision affichage réels	32	Application 4D	-
TCP_NODELAY	33	Application 4D (*)	-
Enreg événements debogage	34	Application 4D (*)	-
Numéro du port client serveur	35	Base de données (*)	Oui
Signature WEDD	36	Base de données (*)	Oui
Inversion des objets	37	Base de données (*)	Oui
Numéro de port HTTPS	39	4D local, 4D Server (*)	Oui
Client Numéro de port HTTPS	40	Tous postes 4D distants (*)	Oui
Mode Unicode	41	Base de données (*)	Oui
Taille mémoire temporaire	42	Application 4D (*)	-
SQL Autocommit	43	Base de données (*)	Oui
Casse caractères moteur SQL	44	Base de données (*)	Oui
Enreg requêtes client	45	Poste 4D distant (*)	-
Chercher par formule serveur	46	Table et process courants	-
Trier par formule serveur	47	Table et process courants	-
Synchro auto dossier Resources	48	Poste 4D distant (*)	-
Jointures CHERCHER PAR FORMULE	49	Process courant (*)	-
Niveau de compression HTTP	50	Application 4D (*)	-
Seuil de compression HTTP	51	Application 4D (*)	-
Timeout connexions inactives	54	Application 4D sauf si valeur négative (***)	-

(*) Dans ce cas, le paramètre table est ignoré s'il est passé.
(**) Ce sélecteur ne peut être utilisé qu'en lecture (voir commande Lire parametre base).
(***) Si le paramètre valeur est négatif, le paramétrage est local au process courant et réinitialisé à la requête suivante.

valeur désigne la valeur du paramètre. La valeur à passer dépend du paramètre que l'on souhaite modifier. Voici les valeurs possibles pour chaque sélecteur :

Sélecteur = 1 (Ratio de tri séq)
Ce sélecteur correspond à un mécanisme ayant été optimisé dans 4D version 11, il est désormais inactivé.

Sélecteur = 2 (Optimisation accès séq)
Ce sélecteur correspond à un mécanisme ayant été optimisé dans 4D version 11, il est désormais inactivé.

Sélecteur = 3 (Ratio valeurs distinctes séq)
Ce sélecteur correspond à un mécanisme ayant été optimisé dans 4D version 11, il est désormais inactivé.

Sélecteur = 4 (Compression index)
Ce sélecteur correspond à un mécanisme ayant été optimisé dans 4D version 11, il est désormais inactivé.

Sélecteur = 5 (Ratio chercher dans sélec séq)
Ce sélecteur correspond à un mécanisme ayant été optimisé dans 4D version 11, il est désormais inactivé.

Sélecteur = 6 (Minimum process Web)
• Valeurs possibles : 0 -> 32 767
• Description : Nombre minimum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 0 (cf. ci-dessous).

Sélecteur = 7 (Maximum process Web)
• Valeurs possibles : 0 -> 32 767
• Description : Nombre maximum de process Web à maintenir en mode sans contexte avec 4D en mode local et 4D Server. Par défaut, la valeur est 10.
Afin que le serveur Web soit réactif, en mode sans contexte, 4D maintient endormis les process Web pendant 5 secondes, et les réutilise pour traiter les éventuelles requêtes HTTP suivantes. En termes de performances, ce principe est en effet plus avantageux que la création d'un nouveau process à chaque requête. Une fois un process Web réutilisé, il est à nouveau endormi pour 5 secondes, sauf si le nombre maximum de process Web est dépassé (auquel cas il est tué). Si aucune requête n'a été attribuée à un process Web durant les 5 secondes, il est tué, sauf si un nombre minimum de process Web a été fixé et est atteint (auquel cas il est à nouveau endormi).
Ces paramètres vous permettent d'ajuster le fonctionnement de votre serveur Web en fonction du nombre de requêtes, de la mémoire disponible, etc.

Sélecteur = 8 (Mode conversion Web)
• Valeurs possibles : 0, 1, 2 ou 3
0 (mode par défaut) = Conversion au format HTML 4.0 si le browser le permet. Sinon, format HTML 3.2 + usage de tableaux.
1 = Conversion au format HTML 2.0 (mode 6.0.x),
2 = Conversion au format HTML 3.2 (mode 6.5),
3 = Conversion au format HTML 4.0 + CSS-P (depuis version 6.5.2).
• Description : Mode de conversion des formulaires 4D pour le Web avec 4D en mode local et 4D Server.
Par défaut, le serveur Web 4D utilise les feuilles de style en cascade (CSS1) pour générer des pages HTML dont l'apparence est très proche de celle des formulaires obtenus dans 4D. Dans le cas des bases créées avec des versions de 4D antérieures à la 6.7, ce fonctionnement peut perturber la conversion correcte des formulaires. Vous pouvez donc modifier le mode de conversion des formulaires.

Ce mode n'est utilisé que pour le process (contexte Web) dans lequel la commande FIXER PARAMETRE BASE est appelée. Elle peut être placée dans la Méthode base Sur connexion Web pour assurer la conversion des formulaires de la base, ou juste avant l'appel d'un formulaire particulier. Hors du mode contextuel ou d'un process Web, la commande est sans effet.

Note : La fonction Lire parametre base admet un sélecteur supplémentaire, Taille cache données (9). Ce paramètre ne peut pas être utilisé avec la commande FIXER PARAMETRE BASE. Pour plus d'informations, reportez-vous à la description de la fonction Lire parametre base.

Sélecteur = 10 (Appels système 4D mode local)
Sélecteur = 11 (Appels système 4D Server)
Sélecteur = 12 (Appels système 4D mode distant)
• Valeurs possibles : pour ces trois sélecteurs, le paramètre valeur est de la forme hexadécimale 0x00aabbcc dans laquelle :
aa = nombre minimum de ticks par appel au système (de 0 à 100 inclus)
bb = nombre maximum de ticks par appel au système (de 0 à 100 inclus)
cc = nombre de ticks entre deux appels au système (de 0 à 20 inclus).
Si une des valeurs est hors de son intervalle, 4D la fixe à son maximum.
Vous pouvez également passer dans valeur une des trois valeurs suivantes, correspondant à un ensemble de réglages standard :
valeur = -1 : priorité maximum allouée à 4D,
valeur = -2 : priorité moyenne allouée à 4D,
valeur = -3 : priorité minimum allouée à 4D.
• Description : Ce sélecteur vous permet de fixer dynamiquement les réglages des appels système de 4D. En fonction du Sélecteur, le gestionnaire d'appels système sera défini pour :
- 4D mode local lorsque la commande est appelée depuis 4D monoposte (sélecteur=10).
- 4D Server lorsque la commande est appelée depuis 4D Server (sélecteur=11).
- 4D mode distant lorsque la commande est appelée depuis 4D connecté à 4D Server (sélecteur=12).

Note : Le fonctionnement du sélecteur 12 (Appels système 4D mode distant) diffère suivant que la commande FIXER PARAMETRE BASE est exécutée sur le poste serveur ou sur un poste client :
- si la commande est exécutée sur le poste serveur, la nouvelle valeur sera appliquée à tous les postes clients se connectant ultérieurement.
- si la commande est exécutée sur un poste client, la nouvelle valeur est appliquée immédiatement au poste client ainsi qu'à tous les postes clients se connectant par la suite au serveur.
Vous pouvez utiliser ce fonctionnement pour mettre en place une gestion dynamique et individualisée de la priorité pour chaque poste client. Le principe consiste à exécuter la commande une première fois sur le poste client à configurer puis une seconde fois sur le poste serveur avec la valeur par défaut, qui sera alors utilisée pour les postes client se connectant par la suite.
Ce fonctionnement est effectif dans 4D à partir des versions 6.8.6, 2003.3 et 2004.

Attention : Paramétrer ces sélecteurs de façon inappropriée peut conduire à une forte dégradation des performances de l'application. Il est conseillé de ne modifier les valeurs par défaut qu'en parfaite connaissance de cause.

Sélecteur = 13 (Timeout 4D Server)
• Description : Ce paramètre permet de modifier la valeur du délai avant déconnexion (timeout) accordé par 4D Server aux postes clients.
Par défaut, la valeur du délai avant déconnexion utilisée par 4D Server est définie dans la page “Client-Serveur/Configuration” de la boîte de dialogue des Préférences, sur le poste serveur.
Le sélecteur Timeout 4D Server vous permet de fixer, à l’aide du paramètre valeur, un nouveau timeout, exprimé en minutes. Cette possibilité permet en particulier d’augmenter la valeur du timeout avant l’exécution sur le poste client d’une opération bloquante et de longue durée, risquant d’entraîner une déconnexion ; par exemple, l’impression d’un grand nombre de pages.
Vous disposez en outre de deux possibilités :
- effectuer une modification globale et permanente : la nouvelle valeur s’applique à tous les process et est stockée dans les préférences de l’application (équivaut à une modification de la valeur dans la boîte de dialogue des Préférences). Pour cela, passez une valeur **positive** dans le paramètre valeur.
- effectuer une modification restreinte et temporaire : la nouvelle valeur ne s’applique qu’au process appelant (les autres process conservant la valeur d’origine), et est abandonnée dès que le serveur reçoit un signe d’activité du poste client — par exemple, dès que l’opération est terminée. Cette possibilité est utile pour gérer les opérations longues initiées par des plug-ins. Pour cela, passez une valeur **négative** dans le paramètre valeur.
Pour définir une connexion “Ouverte en permanence”, passez 0 dans valeur.
Reportez-vous à l’exemple (1).

Sélecteur = 14 (Timeout 4D mode distant)
• Description : Ce paramètre permet de modifier la valeur du délai avant déconnexion (timeout) accordé par le poste 4D distant au poste 4D Server.
Par défaut, la valeur du délai avant déconnexion utilisée par 4D en mode distant est définie dans la page “Client-Serveur/Configuration” de la boîte de dialogue des Préférences, sur le poste distant.
Pour plus d’informations sur le fonctionnement de ce sélecteur, reportez-vous ci-dessus à la description du sélecteur Timeout 4D Server (13).
Le sélecteur Timeout 4D mode distant est à utiliser dans des cas très spécifiques.

Sélecteur = 15 (Numéro du port)
• Description : Ce paramètre permet de modifier par programmation le numéro du port TCP utilisé par le serveur Web 4D avec 4D en mode local et 4D Server. Par défaut, la valeur est 80.
Le numéro de port TCP est défini dans la page “Web/Configuration” de la boîte de dialogue des Préférences. Vous pouvez utiliser les constantes du thème “Numéros de port TCP” pour le paramètre valeur.

Le sélecteur Numéro du port est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d'accès au mode Développement). Pour plus d'informations sur le numéro de port TCP, reportez-vous à la section Paramétrages du serveur Web.

Sélecteur = 16 (Adresse IP d'écoute)
• Description : Ce paramètre permet d'indiquer par programmation l'adresse IP sur laquelle le serveur Web doit recevoir les requêtes HTTP avec 4D en mode local et 4D Server. Par défaut, aucune adresse particulière n'est spécifiée (valeur = 0). Ce paramètre est défini dans les Préférences de la base.
Le sélecteur Adresse IP d'écoute est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop (pas d'accès au mode Développement).
Passez dans le paramètre valeur l'adresse IP sous forme hexadécimale. Ainsi, pour désigner une adresse du type "a.b.c.d", le code sera de la forme :

```
C_ENTIER LONG($addr)
$addr:=($a<<24)|($b<<16)|($c<<8)|$d
FIXER PARAMETRE BASE(Adresse IP d'écoute;$addr)
```

Reportez-vous également à l'exemple (2). Pour plus d'informations sur la désignation de l'adresse IP, reportez-vous à la section Paramétrages du Serveur Web.

Sélecteur = 17 (Jeu de caractères)
• Description : Ce paramètre permet d'indiquer par programmation le jeu de caractères que le serveur Web 4D (avec 4D en mode local et 4D Server) doit utiliser pour communiquer avec les navigateurs Web qui se connectent à la base. La valeur par défaut dépend de la langue du système d'exploitation.
Ce paramètre est défini dans les Préférences de la base. Le sélecteur Jeu de caractères est utile dans le cadre de serveurs Web 4D compilés et fusionnés avec 4D Desktop(pas d'accès au mode Développement).
• Valeurs : Les valeurs possibles dépendent du mode d'exécution de la base relatif au jeu de caractères.
Mode Unicode : Lorsque l'application est exécutée en mode Unicode, les valeurs à passer pour ce paramètre sont des identifiants de jeux de caractères (*MIBEnum*, identifiants définis par l'IANA, cf. http://www.iana.org/assignments/character-sets). Voici la liste des identifiants correspondant aux jeux de caractères pris en charge par le serveur Web de 4D :
4 = ISO-8859-1
12 = ISO-8859-9
13 = ISO-8859-10
17 = ShiftJIS
2026 = Big5
38 = euc-kr
106 = UTF-8
2250 = Windows-1250
2251 = Windows-1251
2253 = Windows-1253
2255 = Windows-1255
2256 = Windows-1256

Note : Dans le cadre du sélecteur Jeu de caractères, la commande Lire parametre base retourne le nom IANA du jeu de caractères dans le paramètre facultatif valeurAlpha.

Mode compatibilité ASCII :
0 : Occidental
1 : Japonais
2 : Chinois
3 : Coréen
4 : Défini par l'utilisateur
5 : Réservé
6 : Europe Centrale
7 : Cyrillique
8 : Arabe
9 : Grec
10 : Hébreu
11 : Turc
12 : Nordique

Note : Pour plus d'informations sur le mode Unicode, reportez-vous à la description du sélecteur 41.

Sélecteur = 18 (Process Web simultanés maxi)
• Valeurs : Vous pouvez passer toute valeur incluse entre 10 et 32 000. La valeur par défaut est 32 000.
• Description : Ce paramètre permet de définir la limite strictement supérieure du nombre de process Web de tout type (contextuels, non contextuels ou appartenant à la réserve de process — cf. à ce sujet le sélecteur 7, Maximum process Web) acceptés par le serveur Web avec 4D en mode local et 4D Server. Lorsque ce nombre limite (moins un) est atteint, 4D ne crée plus de nouveau process et retourne le message "Serveur non disponible" (statut HTTP 503 - Service Unavailable) à toute nouvelle requête.
Ce paramètre permet de prévenir la saturation du serveur Web 4D pouvant se produire lors d'un envoi massif de requêtes ou d'une demande excessive de création de contextes. Il peut également être défini dans la boîte de dialogue des Préférences (cf. section Paramétrages du serveur Web).
En théorie, le nombre maximum de process Web est le résultat de la division Mémoire disponible / Taille de la pile d'un process Web. Une autre solution consiste à visualiser les informations sur les process Web affichées dans l'Exporateur d'exécution : le nombre courant de process Web et le nombre maximum atteint depuis le démarrage du serveur Web sont indiqués.

Note : Si vous passez une valeur inférieure à la limite supérieure de la "réserve" de process (sélecteur 7), cette limite est réduite à la valeur du sélecteur 18. Si nécessaire, la limite inférieure de la réserve (sélecteur 6) est également ajustée.

Sélecteur = 19 (Client Minimum process Web)
Sélecteur = 20 (Client Maximum process Web)
Sélecteur = 21 (Client Taille max requêtes Web)
Sélecteur = 22 (Client Numéro de port)
Sélecteur = 23 (Client Adresse IP d'écoute)
Sélecteur = 24 (Client Jeu de caractères)
Sélecteur = 25 (Client Process Web simultanés maxi)
• Valeurs possibles : Identiques à celles des sélecteurs 4D en mode local ou 4D Server correspondants (cf. sélecteurs 6 à 8, 15 à 18 et 27).
• Description : Ces sélecteurs permettent de spécifier les paramètres de fonctionnement des postes 4D Client utilisés en tant que serveurs Web.
Les valeurs définies via ces sélecteurs sont appliquées à tous les postes 4D Client utilisés comme serveurs Web. Si vous souhaitez définir des valeurs pour certains postes 4D Client uniquement, utilisez la boîte de dialogue des Préférences de 4D Client.

Sélecteur = 26 (Mode écriture cache)
Ce sélecteur correspond à un mécanisme ayant été optimisé dans 4D version 11, il est désormais inactivé.

Sélecteur = 27 (Taille maximum requêtes Web)
• Valeurs possibles : 500 000 à 2 147 483 648.
• Description : Taille maximale (en octets) des requêtes HTTP entrantes (POST) que le serveur Web est autorisé à traiter. Par défaut, la valeur est 2 000 000, c'est-à-dire un peu moins de 2 Mo.
La valeur maximale (2 147 483 648) signifie en pratique qu'aucune limite n'est fixée.
Ce paramétrage permet d'empêcher la saturation du serveur Web à cause de requêtes entrantes de trop grande taille. Lorsqu'une requête atteint la limite, le serveur Web de 4D la refuse.

Sélecteur = 28 (Enreg requêtes 4D Server)
• Valeurs possibles : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier).
• Description : Démarrage ou arrêt de l'enregistrement des requêtes standard reçues par 4D Server (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement de requêtes).
4D Server vous permet d'enregistrer dans un fichier d'historique chaque requête reçue par le poste serveur. Lorsque ce mécanisme est activé, le fichier d'historique est créé à côté du fichier de structure de la base. Son nom est "4DRequestsLogN", où N est le numéro séquentiel de l'historique. Une fois que le fichier atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre valeur.
Ce fichier texte stocke dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, utilisateur, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques. Elles peuvent être importées par exemple dans un logiciel tableur afin d'être traitées.

Note : Il est possible d'activer ou d'inactiver manuellement l'enregistrement des requêtes via le raccourci **Ctrl+Alt+L** sous Windows ou **Commande+Option+L** sous Mac OS.

Sélecteur = 29 (Enreg requêtes Web)
• Valeurs possibles : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF.
• Description : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par le serveur Web de 4D en mode local ou de 4D Server. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes).
L'historique des requêtes Web est stocké sous la forme d'un fichier texte nommé "logweb.txt" automatiquement placé dans le dossier Logs de la base, à côté du fichier de structure. Le format de ce fichier est déterminé par la valeur que vous passez. Pour plus d'informations sur les différents formats d'historique des requêtes, reportez-vous à la section Informations sur le site Web. L'activation de ce fichier peut également être définie dans la page "Web/Avancé" des Préférences de 4D.

ATTENTION : Les formats 3 et 4 sont des formats personnalisés, dont le contenu doit être défini au préalable dans les Préférences de l'application, page "Web/Format du journal". Si vous utilisez l'un de ces formats sans que ses champs n'aient été sélectionnés, le fichier des requêtes n'est pas généré.

Sélecteur = 30 (Client Enreg requêtes Web)
• Valeurs possibles : 0 = Ne pas enregistrer (défaut), 1 = Enregistrer au format CLF, 2 = Enregistrer au format DLF, 3 = Enregistrer au format ELF, 4 = Enregistrer au format WLF.
• Description : Démarrage ou arrêt de l'enregistrement des requêtes Web reçues par les serveurs Web de tous les postes clients. Par défaut, la valeur est 0 (pas d'enregistrement des requêtes).
Le fonctionnement de ce sélecteur est identique à celui du sélecteur 29 ; il s'applique toutefois à tous les postes 4D Client utilisés en tant que serveurs Web. Le fichier est automatiquement placé dans le sous-dossier Logs du dossier base 4D client (dossier de cache). Si vous souhaitez définir des valeurs pour certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D Client.

Sélecteur = 31 (Numéro automatique table)
• Valeurs possibles : Toute valeur de type entier long.
• Description : Ce sélecteur permet de modifier ou de lire le numéro unique courant des enregistrements de la table passée en paramètre. "Numéro courant" signifie "dernier numéro utilisé" : si vous modifiez cette valeur à l'aide de FIXER PARAMETRE BASE, le prochain enregistrement sera créé avec comme numéro la valeur passée + 1. Ce nouveau numéro est, lui, retourné par la commande Numerotation automatique ainsi que dans tout champ de la table auquel la propriété "Incrémentation auto" a été affectée en Structure ou via le SQL.
Par défaut, le numéro unique est défini par 4D et correspond à l'ordre de création des enregistrements.
Pour des informations supplémentaires, reportez-vous à la documentation de la commande Numerotation automatique.

Sélecteur = 32 (Précision affichage réels)
• Valeurs possibles : Toute valeur de type entier long positif.
• Description : Ce sélecteur permet de modifier ou de lire le nombre de chiffres non significatifs tronqués à partir de la droite par l'algorithme d'affichage des réels à l'écran. Cette valeur est définie pour l'application et la session courantes.
Par défaut, la valeur de cette option est 4. La valeur 0 indique que la valeur par défaut est utilisée et que le paramètre n'a pas été modifié durant la session.
Pour des raisons historiques, 4D travaille avec des nombres réels stockés sur 10 octets et les convertit sur 8 octets lors de leur traitement (cf. section Affichage des nombres réels). Ce principe est entièrement transparent et n'affecte pas les calculs, toutefois certains résultats peuvent ne pas donner à l'affichage le résultat escompté. Par exemple, l'opération 4,1-4,09 affiche comme résultat 0,009999999999999780000, mais la recherche de 0,01 retrouve la bonne valeur.
Voici comment 4D procède pour afficher un réel : prenons par exemple la valeur 8,97499999999996158 obtenue par un calcul (la valeur 8,975 étant normalement le résultat attendu). L'algorithme permettant d'arrondir au plus juste enlève par défaut les 4 derniers chiffres (6158) puis vérifie si le dernier chiffre restant est un 0 ou un 9. Si c'est 0, l'algorithme remonte jusqu'au premier 0 et supprime tous les 0. Si la valeur est 9, l'algorithme remonte jusqu'au premier 9 et arrondit la partie décimale à la valeur supérieure. Dans notre exemple, la valeur 8,97499999999996158 se transforme donc en 8,975.
Il peut arriver que certains résultats finissent par 5 chiffres non significatifs, comme par exemple 8,9749999999999986158. Dans ce cas, l'algorithme ne pourra pas arrondir correctement la valeur car il trouvera 8 comme valeur et ne fera alors rien.
Vous pouvez souhaiter que l'algorithme de précision tronque plus ou moins de chiffres en fonction des spécificités de votre base. Dans ce cas, passez une valeur personnalisée. Hormis pour le zéro (choix de la valeur interne de 4D), cette valeur indiquera le nombre de chiffres tronqués par l'algorithme de précision.
Rappelons que ce paramétrage n'influe que sur l'affichage des numériques, non sur leur traitement interne.

Sélecteur = 33 (TCP_NODELAY)
• Valeurs possibles : 0 ou 1 (0 = inactiver, 1 = activer)
• Description : Activation ou inactivation de l'option réseau TCP_NODELAY. Cette option, interne au protocole TCP/IP, contrôle un mécanisme d'optimisation des communications réseau. Elle peut être fixée séparément pour le poste serveur et les postes clients. Par défaut, la valeur est 1 (option activée) sur tous les postes (serveur et clients).
Dans des cas spécifiques, en particulier dans le cadre de connexions client/serveur par ADSL ou par réseau privé virtuel (VPN), la désactivation de cette option peut améliorer sensiblement les performances de l'application. Cette opération doit être effectuée avec précaution et doit s'accompagner de tests de charge dans différentes configurations client/serveur.
Lorsque vous modifiez cette valeur, il est nécessaire de redémarrer l'application afin que la nouvelle valeur soit prise en compte.

Sélecteur = 34 (Enreg événements debogage)
• Valeurs possibles : 0, 1 ou 2 (0 = ne pas enregistrer, 1 = enregistrer, 2 = enregistrer en mode détaillé)

• Description : Démarrage ou arrêt de l'enregistrement séquentiel des événements de programmation de 4D, destiné au débogage de l'application. Par défaut, la valeur est 0 (pas d'enregistrement des événements).
Lorsque ce mode est activé, diverses informations sont enregistrées, notamment :
- pour chaque événement, le nombre de millisecondes depuis la création du fichier et le numéro de process (*[n]*)
- l'exécution de chaque commande 4D (*cmd*) et chaque appel de plug-in (*plugInName*) ; le niveau de la pile est indiqué dans ce cas *(n)* ,
- chaque appel de méthode projet (*meth*), de méthode objet (*obj*) et de méthode formulaire (*form*).
- lorsque le mode détaillé est activé (valeur = 2), des informations supplémentaires relatives aux plug-ins sont enregistrées : événements dans les zones de plug-ins (*EventCode*) et appels de 4D par les plug-ins (*externCall*).
Les événements sont stockés dans un fichier nommé "4DDebugLog.txt" automatiquement placé dans le sous-dossier Logs de la base, à côté du fichier de structure. Chaque événement est systématiquement inscrit dans le fichier avant son exécution, ce qui garantit sa présence dans le fichier même lorsque l'application quitte inopinément. A noter que le fichier est effacé et réécrit à chaque lancement de l'application.
L'option peut être activée dans tout type d'application 4D (4D tous modes, 4D Server, 4D Volume Desktop), en interprété ou en compilé.

Note : Cette option est proposée uniquement à des fins de débogage, elle ne doit pas être utilisée en production car elle peut entraîner une dégradation des performances de l'application ainsi que la saturation du disque dur.

Sélecteur = 35 (Numéro du port client serveur)
• Valeurs possibles : 0 à 65535
• Description : Ce paramètre permet de modifier par programmation le numéro de port TCP sur lequel 4D Server publie la base de données (à destination des postes 4D Client). Par défaut, la valeur est 19813.
La personnalisation de cette valeur permet d'utiliser plusieurs applications 4D client-serveur sur la même machine avec le protocole TCP ; dans ce cas, vous devez spécifier un numéro de port différent pour chaque application.
La valeur est stockée dans le fichier de structure de la base. Elle peut être définie avec 4D en mode local mais n'est prise en compte qu'en configuration client-serveur.
Lorsque vous modifiez cette valeur, il est nécessaire de redémarrer le poste serveur afin que la nouvelle valeur soit prise en compte.

Sélecteur = 36 (Signature WEDD)
• Valeurs possibles : Chaîne de 1 à 255 caractères.
• Description : Ce paramètre permet de modifier la signature WEDD de la base de données ouverte (fichier de structure et fichier de données). Par défaut, cette signature est vide (le paramètre n'est pas défini). A noter que la signature tient compte de la casse des caractères.
La signature WEDD permet d'associer un fichier de structure à un fichier de données. Un fichier de structure contenant une signature WEDD ne pourra être ouvert qu'avec un fichier de données contenant la même signature WEDD et inversement. Pour plus d'informations sur la signature WEDD, reportez-vous au manuel *Mode Développement*.

La définition de cette valeur par programmation facilite la diffusion de mises à jour d'applications comportant une signature personnalisée.

Note : Lorsque vous utilisez ce sélecteur avec la commande Lire parametre base, la chaîne définie comme signature WEDD est retournée dans le paramètre facultatif valeurAlpha et la commande retourne 0.

Sélecteur = 37 (Inversion des objets)
• Valeurs possibles : 0, 1 ou 2 (0 = mode désactivé, 1 = mode automatique, 2 = mode activé).
• Description : Configuration du mode "inversion des objets". Ce mode permet d'inverser en mode Application les formulaires, objets, menus, etc. lorsque la base est affichée sous Windows dans une langue de droite à gauche. Ce mode peut également être configuré dans la page Base de données/Script Manager des Préférences de l'application.
- La valeur 0 indique que le mode n'est jamais activé, quelle que soit la configuration du système (correspond à la valeur Non dans les Préférences).
- La valeur 1 indique que le mode est activé ou non en fonction de la configuration du système (correspond à la valeur Automatique dans les Préférences).
- La valeur 2 indique que le mode est activé, quelle que soit la configuration du système (correspond à la valeur Oui dans les Préférences).
Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Sélecteur = 39 (Numéro de port HTTPS)
• Valeurs possibles : 0 à 65535
• Description : Ce sélecteur permet de modifier par programmation le numéro du port TCP utilisé par le serveur Web de 4D en mode local et de 4D Server pour les connexions sécurisées via SSL (protocole HTTPS). Le numéro de port HTTPS est défini dans la page "Web/Configuration" de la boîte de dialogue des Préférences. Pour plus d'informations, reportez-vous à la section Paramétrages du serveur Web.
Par défaut, la valeur est 443 (valeur standard). Vous pouvez utiliser les constantes du thème "Numéros de port TCP" pour le paramètre valeur.

Sélecteur = 40 (Client Numéro de port HTTPS)
• Valeurs possibles : 0 à 65535
• Description : Ce paramètre permet de modifier par programmation le numéro du port TCP utilisé par les serveurs Web des postes clients pour les connexions sécurisées via SSL (protocole HTTPS). Par défaut, la valeur est 443 (valeur standard).
Le fonctionnement de ce sélecteur est identique à celui du sélecteur 39 ; il s'applique toutefois à tous les postes 4D Client utilisés en tant que serveurs Web. Si vous souhaitez modifier la valeur de certains postes clients uniquement, utilisez la boîte de dialogue des Préférences de 4D Client.

Sélecteur = 41 (Mode Unicode)
• Valeurs possibles : 0 (mode compatibilité) ou 1 (mode Unicode)
• Description : Mode d'exécution courant de la base, relatif au jeu de caractères. 4D prend en charge le jeu de caractères Unicode mais peut fonctionner en mode "compatibilité" (basé sur le jeu de caractères ASCII Mac). Par défaut, les bases de données converties sont exécutées en mode compatibilité (0) et les bases créées à partir de la version 11 sont exécutées en mode Unicode. Le mode d'exécution est contrôlé via une option des préférences et peut également être lu ou (à des fins de test) modifié via ce sélecteur. La modification de cette option nécessite le redémarrage de la base pour être prise en compte. A noter que, dans le cadre d'un composant, il n'est pas possible de modifier cette valeur, mais uniquement de la lire.

Sélecteur = 42 (Taille mémoire temporaire)
• Valeurs possibles : Entier long positif > 50 000 000
• Description : Taille de la mémoire temporaire exprimée en octets. Par défaut, cette taille est de 50 000 000 (50 Mo).
4D utilise une mémoire temporaire spéciale dédiée aux opérations d'indexation et de tri. Cette mémoire a pour but de préserver la mémoire cache "classique" lors d'opérations massives. Lorsque la taille maximale de la mémoire temporaire est atteinte (cas critique), 4D interrompt automatiquement la dernière opération demandée afin que cela ne pénalise pas les autres traitements. Dans la plupart des cas, la configuration par défaut de la mémoire temporaire sera parfaitement suffisante. Toutefois, dans certaines applications spécifiques effectuant des tris ou des indexations de façon très intensive, l'augmentation de la taille de cette mémoire pourra contribuer à améliorer sensiblement les performances. La valeur idéale sera à déterminer de façon empirique en fonction des spécificités de l'application et résultera généralement de tests en volumétrie réelle.

Sélecteur = 43 (SQL Autocommit)
• Valeurs possibles : 0 (désactivation) ou 1 (activation)
• Description : Activation ou désactivation du mode SQL auto-commit. Par défaut, la valeur est 0 (mode désactivé)
Le mode auto-commit permet de renforcer l'intégrité référentielle de la base. Lorsque ce mode est actif, les requêtes SELECT, INSERT, UPDATE et DELETE (SIUD) sont automatiquement incluses dans des transactions lorsqu'elles sont exécutées en-dehors de toute transaction. Ce mode peut également être défini dans les préférences de la base.

Sélecteur = 44 (Casse caractères moteur SQL)
• Valeurs possibles : 0 (casse non prise en compte) ou 1 (casse prise en compte)
• Description : Activation ou désactivation de la prise en compte de la casse des caractères pour les comparaisons de chaînes effectuées par le moteur SQL.
Par défaut, la valeur est 1 (casse prise en compte) : le moteur SQL établit une différence entre les majuscules et les minuscules lors des comparaisons de chaînes (tris et recherches). Par exemple "ABC"= "ABC" mais "ABC" # "Abc". Dans certains cas, par exemple pour aligner le fonctionnement du moteur SQL sur celui du moteur 4D, vous pourrez souhaiter que les comparaisons de chaînes ne tiennent pas compte de la casse ("ABC"="Abc").
Cette option peut également être définie dans la page SQL/Configuration des Préférences de l'application.

Sélecteur = 45 (Enreg requêtes client)
• Valeurs possibles : 0 ou de 1 à N (0 = ne pas enregistrer, 1 à N = numéro séquentiel, accolé au nom du fichier).
• Description : Démarrage ou arrêt de l'enregistrement des requêtes standard effectuées par le poste client 4D ayant exécuté la commande (hors requêtes Web). Par défaut, la valeur est 0 (pas d'enregistrement des requêtes).
4D vous permet d'enregistrer l'historique des requêtes effectuées par le poste client. Lorsque ce mécanisme est activé, deux fichiers sont créés sur le poste client, dans le sous-dossier Logs du dossier local de la base. Il sont nommés *4DRequestsLog_N* et *4DRequestsLog_ProcessInfo_N* où N est le numéro séquentiel de l'historique. Une fois que le fichier *4DRequestsLog* atteint une taille de 10 Mo, il est refermé et un nouveau fichier est généré, avec un numéro séquentiel incrémenté. Si un fichier du même nom existe déjà, il est directement remplacé. Vous pouvez définir le numéro de départ de la séquence à l'aide du paramètre valeur.
Ces fichiers texte stockent dans un format tabulé simple diverses informations concernant chaque requête : heure, numéro de process, taille de la requête, durée de traitement, etc. Ces informations sont particulièrement utiles en phase de mise au point de l'application ou à des fins statistiques.

Sélecteur = 46 (Chercher par formule serveur)
• Valeurs possibles : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur)
• Description : Emplacement de l'exécution des commandes CHERCHER PAR FORMULE et CHERCHER PAR FORMULE DANS SELECTION pour la table passée en paramètre.
Dans le cadre de l'exploitation d'une base en client-serveur, les commandes de recherche "par formule" peuvent exécutées soit sur le serveur soit sur le client :
- dans les bases de données créées avec 4D v11 SQL, ces commandes sont exécutées sur le serveur.
- dans les bases de données converties, ces commandes sont exécutées sur le client, comme dans les versions précédentes de 4D.
- dans les bases de données converties, une préférence spécifique (page Développement/ Compatibilité) permet de modifier globalement le lieu d'exécution de ces commandes.
Cette différence de lieu d'exécution influe sur les performances de l'application (l'exécution sur le serveur est généralement plus rapide) mais également sur la programmation. En effet, la valeur des composantes de la formule (notamment les variables appelées via une méthode) diffère suivant le contexte d'exécution. Vous pouvez utiliser ce sélecteur pour adapter ponctuellement le fonctionnement de votre application.
Si vous passez 0 dans le paramètre valeur, l'emplacement d'exécution des commandes de recherche "par formule" dépendra de la configuration de la base : dans les bases créées avec 4D v11 SQL, les commandes seront exécutées sur le serveur. Dans les bases converties, elles seront exécutées sur le client ou le serveur en fonction des préférences de la base. Passez 1 ou 2 dans valeur pour "forcer" l'exécution des commandes respectivement sur le client ou sur le serveur.
Reportez-vous à l'exemple 4.

Sélecteur = 47 (Trier par formule serveur)
• Valeurs : 0 (utiliser le paramétrage de la base), 1 (exécuter sur le client) ou 2 (exécuter sur le serveur)
• Description : Emplacement de l'exécution de la commande TRIER PAR FORMULE pour la table passée en paramètre.
Dans le cadre de l'exploitation d'une base en client-serveur, la commande TRIER PAR FORMULE peut être exécutée soit sur le serveur soit sur le client. Ce sélecteur permet de définir l'emplacement de l'exécution de cette commande (serveur ou client). Ce mode peut également être défini dans les préférences de la base. Pour plus d'informations, reportez-vous à la description du sélecteur 46, Chercher par formule serveur.

Sélecteur = 48 (Synchro auto dossier Resources)
• Valeurs : 0 (pas de synchronisation), 1 (synchronisation auto) ou 2 (demander).
• Description : Mode de synchronisation dynamique du dossier Resources du poste client 4D ayant exécuté la commande avec celui du serveur.
Lorsque le contenu du dossier Resources sur le serveur a été modifié ou qu'une demande de synchronisation a été émise (via l'explorateur de ressources ou suite à l'exécution de la commande NOTIFIER MODIFICATION DOSSIER RESOURCES), le serveur notifie les clients connectés.
Trois modes de synchronisation sont alors possibles côté client. Le sélecteur Synchro auto dossier Resources vous permet de définir le mode à utiliser pour le poste client et la session courante :
- valeur = 0 (valeur par défaut) : pas de synchronisation dynamique (la demande de synchronisation est ignorée)
- valeur = 1 : synchronisation dynamique automatique
- valeur = 2 : affichage d'une boîte de dialogue sur les postes clients, avec possibilité d'effectuer ou de refuser la synchronisation.
Le mode de synchronisation peut également être défini globalement dans les Préférences de l'application.

Sélecteur = 49 (Jointures CHERCHER PAR FORMULE)
• Valeurs possibles : 0 (utiliser paramétrages de la base), 1 (toujours utiliser les liens auto) ou 2 (utiliser les jointures SQL si possible).
• Description : Mode de fonctionnement des commandes CHERCHER PAR FORMULE et CHERCHER PAR FORMULE DANS SELECTION relatif à l'utilisation de "jointures SQL".
Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, ces commandes effectuent des jointures sur le modèle des jointures SQL. Ce mécanisme permet de modifier la sélection d'une table en fonction d'une recherche effectuée sur une autre table sans que les tables soient reliées par un lien automatique (condition nécessaire dans les versions précédentes de 4D).
Le sélecteur Jointures CHERCHER PAR FORMULE vous permet de définir le mode de fonctionnement des commandes de recherche par formule pour le process courant :
- valeur = 0 : utiliser les paramètres courants de la base (valeur par défaut). Dans les bases de données créées à compter de la version 11.2 de 4D v11 SQL, les "jointures SQL" sont toujours activées pour les recherches par formule. Dans les bases de données converties, ce mécanisme est inactivé par défaut pour des raisons des compatibilité mais peut être mis en oeuvre via une préférence.

- valeur = 1 : toujours utiliser les liens auto (= fonctionnement des versions précédentes de 4D). Dans ce mode, un lien est nécessaire pour définir la sélection d'une table en fonction de recherches effectuées dans une autre table. 4D n'effectue pas de "jointures SQL".
- valeur = 2 : utiliser les jointures SQL si possible (= fonctionnement par défaut des bases créées en version 11.2 et suivantes de 4D v11 SQL). Dans ce mode, 4D établit des "jointures SQL" pour les recherches par formule lorsque la formule s'y prête (à deux exceptions près, voir la description de la commande CHERCHER PAR FORMULE ou CHERCHER PAR FORMULE DANS SELECTION.

Sélecteur = 50 (Niveau de compression HTTP)
• Valeurs possibles : 1 à 9 (1 = plus rapide, 9 = plus compressé) ou -1 = meilleur compromis.
• Description : Fixe le niveau de compression pour tous les échanges HTTP compressés effectués pour des Web Services (requêtes client ou réponses serveur). Les échanges compressés sont une optimisation que vous pouvez mettre en oeuvre lorsque vous faites communiquer deux applications 4D via des Web services (cf. commande FIXER OPTION WEB SERVICE). Ce sélecteur vous permet d'optimiser les échanges en privilégiant la rapidité d'exécution (compression moindre) ou la quantité de compression (vitesse moindre). Le choix d'une valeur dépend de la taille et de la nature des données échangées. Passez de 1 à 9 dans le paramètre valeur, 1 étant la compression la plus rapide et 9 la plus élevée. Vous pouvez également passer -1 pour obtenir un compromis entre vitesse et taux de compression. Par défaut le niveau de compression est 1 (compression rapide).

Sélecteur = 51 (Seuil de compression HTTP)
• Valeurs possibles : Toute valeur de type Entier long.
• Description : Dans le cadre d'échanges Web Services inter-4D en mode optimisé (cf. ci-dessus), fixe le seuil de taille de requête au-dessous de laquelle les échanges ne doivent pas être compressés. Ce paramétrage est utile pour éviter de perdre du temps machine à compresser les petits échanges.
Passez dans valeur une taille exprimée en octets. Par défaut, le seul de compression est fixé à 1024 octets.

Sélecteur = 54 (Timeout connexions inactives)
• Valeurs : Valeur entière exprimant une durée en secondes. La valeur peut être positive (nouvelles connexions) ou négative (connexions existantes). Par défaut, la valeur est 0 (pas de timeout).
• Description : Ce paramètre permet de définir, côté serveur, le délai maximum d'inactivité (timeout) des connexions au moteur de base de données et au moteur SQL de 4D. Lorsqu'une connexion inactive atteint ce délai, elle est automatiquement mise en veille, ce qui se traduit par le gel de la session client/serveur et la fermeture du socket réseau. Ce fonctionnement est entièrement transparent pour l'utilisateur : dès qu'il y a reprise d'activité sur la connexion mise en veille, le socket est automatiquement rouvert et la session client/serveur restaurée.
Ce paramétrage permet, d'une part, d'économiser des ressources sur le serveur : les connexions mises en veille referment le socket et libèrent un processus sur le serveur. D'autre part, il permet d'éviter les pertes de connexions dues aux fermetures par les pare-feux des sockets inactifs. La valeur de timeout des connexions inactives doit pour cela être inférieure à celle du pare-feu.

Si vous passez une valeur positive dans valeur, elle s'applique à toutes les nouvelles connexions dans tous les process. Si vous passez une valeur négative, elle s'applique aux connexions ouvertes dans le process courant. Si vous passez 0, les connexions inactives ne sont pas soumises à un timeout.

Exemples

(1) L'instruction suivante permet d'anticiper un éventuel problème de **timeout** :

```
    `Augmentation du timeout à 3 heures pour le process courant
FIXER PARAMETRE BASE(Timeout 4D Server;-60*3)
    `Exécution d'une opération longue hors du contrôle de 4D
...
WR MAILING (LaZone;3;0)
...
```

(2) L'**adresse IP** 192.193.194.195 sera fixée avec l'instruction suivante :

```
FIXER PARAMETRE BASE(Adresse IP d'écoute;0xC0C1C2C3)
```

(3) Ce code permet de permuter le jeu de caractères courant et de redémarrer la base :

```
Mode_unicode_courant:=Lire parametre base(Mode Unicode)
FIXER PARAMETRE BASE(Mode Unicode;1-Mode_unicode_courant)
OUVRIR FICHIER DONNEES(Fichier donnees)
```

(4) Cet exemple force temporairement l'exécution sur le client d'une commande de recherche par formule :

```
valCourante:= Lire parametre base([table1];Chercher par formule serveur)
    `Stocker le paramétrage courant
FIXER PARAMETRE BASE([table1];Chercher par formule serveur;1)
    `Forcer l'exécution sur le client
CHERCHER PAR FORMULE([table1];maformule)
FIXER PARAMETRE BASE([table1];Chercher par formule serveur;valCourante)
    `Rétablir le paramétrage courant
```

Référence

CHERCHER DANS SELECTION, Lire parametre base.

GENERER APPLICATION {(nomProjet)}

Paramètre	Type		Description
nomProjet	Chaîne	→	Chemin d'accès complet du projet à utiliser

Description

La commande GENERER APPLICATION lance le processus de génération d'application en prenant en compte les paramètres définis dans le projet d'application courant ou le projet d'application désigné par le paramètre nomProjet.

Un projet d'application est un fichier XML contenant tous les paramétrages utilisés pour générer une application. La plupart de ces paramétrages sont visibles dans la boîte de dialogue du Générateur d'application (pour plus d'informations sur le Générateur d'application, reportez-vous au manuel *Mode Développement* de 4D).
Par défaut, 4D crée pour chaque base de données un projet d'application par défaut nommé "buildapp.xml" et placé dans le sous-dossier BuildApp du dossier Preferences de la base.

Si la base n'a pas été compilée ou si le code compilé n'est pas à jour, la commande lance au préalable le processus de compilation. Dans ce cas, la fenêtre du compilateur n'apparaît pas (sauf en cas d'erreur), seule une barre de progression est affichée.

Si vous ne passez pas le paramètre facultatif nomProjet, la commande affiche une boîte de dialogue standard d'ouverture de document, vous permettant de désigner un fichier de projet. La variable système *Document* contiendra le chemin d'accès complet du fichier sélectionné. Si vous passez le chemin d'accès et le nom d'un fichier XML de projet d'application valide (extension ".xml"), la commande utilisera les paramètres définis dans le fichier. Pour plus d'informations sur la structure et les clés utilisables dans un fichier XML de projet d'application, reportez-vous au manuel *Clés XML de 4D*.

Exemple

Génération de deux applications dans une seule méthode :

```
GENERER APPLICATION("c:\\dossier\\projets\\monprojet1.xml")
Si (OK=1)
    GENERER APPLICATION("c:\\dossier\\projets\\monprojet2.xml")
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si la commande a été correctement exécutée, sinon elle prend la valeur 0. La variable système Document prend le chemin d'accès complet du fichier de projet ouvert.

Gestion des erreurs

Si la commande échoue, une erreur est générée, que vous pouvez intercepter à l'aide de la commande APPELER SUR ERREUR.

LIRE INFORMATIONS SERIALISATION (clé; utilisateur; société; connectés; maxUtilisateurs)

Paramètre	Type		Description
clé	Entier long	←	Clé unique du produit (crypté)
utilisateur	Alpha	←	Nom enregistré
société	Alpha	←	Organisation enregistrée
connectés	Entier long	←	Nombre d'utilisateurs connectés
maxUtilisateurs	Entier long	←	Nombre maximum d'utilisateurs

Description

La commande LIRE INFORMATIONS SERIALISATION retourne diverses informations relatives à la sérialisation de l'application 4D courante.

• clé : identifiant unique du produit installé. Ce numéro unique correspond à une seule application 4D (4D Server, 4D en mode local, 4D Desktop, etc.) installée sur un seul poste. Bien entendu, ce numéro est crypté.
• utilisateur : Nom de l'utilisateur de l'application, tel qu'il a été saisi au moment de l'installation.
• société : Nom de la société ou de l'organisation à laquelle appartient l'utilisateur, tel qu'il a été saisi au moment de l'installation.
• connectés : Nombre d'utilisateurs connectés au moment de l'exécution de la commande.
• maxUtilisateurs : Nombre maximal d'utilisateurs pouvant se connecter simultanément.

Note : Les deux derniers paramètres retournent toujours 1 pour les versions monopostes de 4D, sauf lorsqu'il s'agit de versions de démonstration, auquel cas ils retournent 0.

La commande LIRE INFORMATIONS SERIALISATION s'inscrit dans le schéma général de protection des composants proposé par 4D.

Les développeurs de composants peuvent, s'ils le souhaitent, lier chaque copie de leur produit à une seule application 4D installée, afin d'empêcher toute copie illicite.

Le principe de fonctionnement du système est le suivant : un utilisateur souhaitant acquérir un composant fournit au développeur sa clé unique — générée à l'aide de la commande LIRE INFORMATIONS SERIALISATION. Cette opération peut, par exemple, être effectuée par l'intermédiaire d'un formulaire "Bon de commande" intégré à la version de démonstration du composant. La clé générée est unique : il n'existe qu'une clé par application 4D installée.

Le développeur du composant peut alors générer son propre numéro de série, en combinant la clé et l'algorithme de cryptage de son choix. Le composant livré comportera une fonction permettant de tester si les informations retournées par LIRE INFORMATIONS SERIALISATION correspondent bien à ce numéro de série. Dans le cas contraire, le composant sera rendu inutilisable.

Note : Les développeurs de plug-ins peuvent également bénéficier de ce système de protection. Pour plus d'informations, reportez-vous à la documentation de *4D Plugin Kit*.

Référence

Lire ID ressource composant.

Lire langue courante base → Chaîne

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Chaîne	←	Langue courante de la base de données

Description

La commande Lire langue courante base retourne la langue courante utilisée par la base de données, exprimée dans la norme définie par la RFC 3066. Typiquement, la commande retourne “fr” pour le français “es” pour l’espagnol, etc. Pour plus d’informations sur cette norme et sur les valeurs retournées par cette commande, reportez-vous au manuel *Mode Développement*.

La langue courante de la base permet de définir le dossier .lproj dans lequel le programme va chercher les éléments localisés de la base de données. 4D détermine automatiquement la langue courante au démarrage de la base en fonction du contenu du dossier Resources et de l’environnement système. Le principe est que 4D charge le premier dossier .lproj de la base correspondant à la langue de référence, dans l’ordre de priorité suivant :
1. Langue du système (sous Mac OS, plusieurs langues peuvent être définies avec un ordre de préférence, 4D utilise ce paramétrage).
2. Langue de l’application 4D.
3. Anglais
4. Première langue trouvée dans le dossier Resources.

Note : Si la base ne contient aucun dossier .lproj, 4D applique l'ordre de priorité suivant : 1. Langue du système, 2. Anglais (si la langue du système n'a pas pu être identifiée).

Lire parametre base

Lire parametre base ({table; }sélecteur{; valeurAlpha}) → Entier long

Paramètre	Type		Description
table	Table	→	Table du paramètre ou Table par défaut si ce paramètre est omis
sélecteur	Entier long	→	Code du paramètre de la base
valeurAlpha	Alpha	←	Valeur alpha du paramètre
Résultat	Entier long	←	Valeur du paramètre

Description

La commande Lire parametre base permet de lire la valeur courante d'un paramètre de la base 4D. Lorsque la valeur du paramètre est une chaîne de caractères, elle est retournée dans le paramètre valeurAlpha.

sélecteur désigne le paramètre de la base à lire. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème “Paramètres de la base” :

Constante	Type	Valeur
Ratio de tri séq	**** *Sélecteur inactivé* ****	
Optimisation accès séq	**** *Sélecteur inactivé* ****	
Ratio valeurs distinctes séq	**** *Sélecteur inactivé* ****	
Compression index	**** *Sélecteur inactivé* ****	
Ratio chercher dans sélec séq	**** *Sélecteur inactivé* ****	
Minimum process Web	Entier long	6
Maximum process Web	Entier long	7
Mode conversion Web	Entier long	8
Taille cache données	Entier long	9
Appels système 4D mode local	Entier long	10
Appels système 4D Server	Entier long	11
Appels système 4D mode distant	Entier long	12
Timeout 4D Server	Entier long	13
Timeout 4D mode distant	Entier long	14
Numéro du port	Entier long	15
Adresse IP d’écoute	Entier long	16
Jeu de caractères	Entier long	17
Process Web simultanés maxi	Entier long	18
Client Minimum process Web	Entier long	19
Client Maximum process Web	Entier long	20

Client Mode conversion Web	Entier long	21
Client Numéro de port	Entier long	22
Client Adresse IP d'écoute	Entier long	23
Client Jeu de caractères	Entier long	24
Client Process Web simultanés maxi	Entier long	25
Mode écriture cache	**** *Sélecteur inactivé* ****	
Taille maximum requêtes	Entier long	27
Enreg requêtes 4D Server	Entier long	28
Enreg requêtes Web	Entier long	29
Client Enreg requêtes Web	Entier long	30
Numéro automatique table	Entier long	31
Précision affichage réels	Entier long	32
TCP_NODELAY	Entier long	33
Enreg événements debogage	Entier long	34
Numéro du port client serveur	Entier long	35
Signature WEDD	Entier long	36
Inversion des objets	Entier long	37
Numéro de port HTTPS	Entier long	39
Client Numéro de port HTTPS	Entier long	40
Mode Unicode	Entier long	41
Taille mémoire temporaire	Entier long	42
SQL Autocommit	Entier long	43
Casse caractères moteur SQL	Entier long	44
Enreg requêtes client	Entier long	45
Chercher par formule serveur	Entier long	46
Trier par formule serveur	Entier long	47
Client Synchro auto Resources	Entier long	48
Jointures CHERCHER PAR FORMULE	Entier long	49
Niveau de compression HTTP	Entier long	50
Seuil de compression HTTP	Entier long	51
Timeout connexions inactives	Entier long	54

Les valeurs pouvant être retournées par cette fonction ainsi que la portée et la conservation des sélecteurs sont détaillées dans la description de la commande FIXER PARAMETRE BASE.

Le sélecteur Taille cache données (9) vous permet d'obtenir la taille courante du cache mémoire utilisé par 4D pour les données. Cette valeur est exprimée en octets.
La taille du cache est issue des paramétrages définis dans la page "Base de données/Gestion des données" des Préférences. La taille réelle allouée au cache dépend de ces paramètres mais également de l'état des ressources mémoire de la machine.
Ce sélecteur vous permet donc de connaître précisément la taille courante de la mémoire allouée au cache par 4D.

Note : La taille du cache de données ne peut pas être fixée par programmation. Autrement dit, il n'est pas possible d'utiliser le sélecteur Taille cache données avec la commande FIXER PARAMETRE BASE.

Lorsque vous utilisez le sélecteur Jeu de caractères (17 ou 24) avec cette commande, le nom IANA du jeu de caractères est retourné dans le paramètre facultatif valeurAlpha.

Lorsque vous utilisez le sélecteur Signature WEDD (36) avec cette commande, la chaîne définie comme signature WEDD est retournée dans le paramètre facultatif valeurAlpha et la commande retourne 0 comme résultat.

Exemples

(1) Cette méthode permet de récupérer les valeurs courantes du minuteur interne de 4D :

```
C_ENTIER LONG($ticksbtwcalls;$maxticks;$minticks;$lparams)
Si (Type application=4D mode local) ` Si nous sommes en 4D local
    $lparams:=Lire parametre base(Appels système 4D mode local)
    $ticksbtwcalls:=$lparams & 0x00ff
    $maxticks:=($lparams>>8) & 0x00ff
    $minticks:=($lparams>>16) & 0x00ff
Fin de si
```

(2) Le sélecteur 16 (Adresse IP d'écoute) permet d'obtenir l'adresse IP sur laquelle le serveur Web 4D reçoit les requêtes HTTP. L'adresse obtenue est de forme hexadécimale. L'exemple suivant permet de décomposer l'adresse IP reçue :

```
C_ENTIER LONG($a;$b;$c;$d)
C_ENTIER LONG($addr)
$addr:=Lire parametre base(Adresse IP d'écoute)
$a:=($addr>>24)&0x000000ff
$b:=($addr>>16)&0x000000ff
$c:=($addr>>8)&0x000000ff
$d:=$addr&0x000000ff
```

Référence

CHERCHER DANS SELECTION, FIXER PARAMETRE BASE, VALEURS DISTINCTES.

LISTE COMPOSANTS (tabComposants)

Paramètre	Type		Description
tabComposants	Tab chaîne	←	Noms des composants

Description

A l'ouverture d'une base, 4D charge les composants valides situés dans le dossier Components situé à côté du fichier de structure. La commande LISTE COMPOSANTS dimensionne et remplit le tableau tabComposants avec les noms des composants chargés par l'application 4D pour la base hôte courante.

Cette commande peut être appelée depuis la base hôte ou depuis un composant. Si la base n'utilise pas de composant, le tableau tabComposants est retourné vide.

Les noms des composants sont les noms des fichiers de structure des bases matrices (.4db, .4dc ou .4dbase). Cette commande permet de mettre en place des architectures et des interfaces modulaires proposant des fonctionnalités supplémentaires en fonction de la présence des composants.

Pour plus d'informations sur les composants 4D, reportez-vous au manuel *Mode Développement*.

LISTE PLUGINS (tabNuméros; tabNoms)

Paramètre	Type		Description
tabNuméros	Tab Entier long	←	Numéros des plug-ins
tabNoms	Tab Alpha	←	Noms des plug-ins

Description

La commande LISTE PLUGINS remplit les tableaux tabNuméros et tabNoms avec les numéros et les noms des plug-ins chargés par l'application 4D et utilisables. Les deux tableaux sont automatiquement dimensionnés et synchronisés par la commande.

Note : Vous pouvez comparer les valeurs retournées dans le tableau tabNuméros avec les constantes du thème “Licence disponible”.

LISTE PLUGINS prend en compte tous les plug-ins, y compris les plug-ins intégrés (par exemple 4D Chart) et les plug-ins des éditeurs tiers.

Référence

ECRIRE ACCES PLUGIN, Licence disponible, Lire acces plugin, LISTE COMPOSANTS.

LISTE SEGMENTS DE DONNEES (segments)

Paramètre	Type		Description
segments	Tableau alpha	←	Noms longs des segments de données de la base

Description

LISTE SEGMENTS DE DONNEES remplit le tableau segments avec les noms complets (chemin d'accès + nom de fichier) des segments de données de la base avec laquelle vous travaillez.

Note de compatibilité : Depuis la version 11 de 4D, les segments de données ne sont plus pris en charge (la taille du fichier de données est désormais illimitée). Cette commande a été conservée par compatibilité. Elle retourne désormais systématiquement un tableau à un élément, contenant le nom long du fichier de données de la base.

Référence

Fichier application, Fichier donnees, Fichier structure.

Mode compile

Mode compile {(*)} → Booléen

Paramètre	Type		Description
*	*	→	Retourner l'information de la base hôte
Résultat	Booléen	←	Mode compilé (Vrai), mode interprété (Faux)

Description

La fonction Mode compile teste si la base tourne en mode compilé (Vrai) ou en mode interprété (Faux).

Le paramètre facultatif * est utile dans le cadre d'une architecture utilisant des composants : il permet de déterminer la base (hôte ou composant) dont vous souhaitez connaître le mode d'exécution.
• Lorsque la commande est appelée depuis un composant :
- si le paramètre * est passé, la commande retourne Vrai ou Faux en fonction du mode d'exécution de la base hôte,
- si le paramètre * n'est pas passé, la commande retourne Vrai ou Faux en fonction du mode d'exécution du composant.
• Lorsque la commande est appelée depuis une méthode d'une base hôte, elle retourne toujours Vrai ou Faux en fonction du mode d'exécution de la base hôte.

Exemple

Dans une de vos méthodes, vous avez placé du code pour déboguer la base lorsque vous êtes en mode interprété. Vous pouvez précéder ce code d'un test qui appelle la fonction Mode compile :

```
` ...
Si (Non(Mode compile))
   ` Mettre du code pour déboguer votre base ici
Fin de si
` ...
```

Référence

APPELER 4D, Indefinie.

NOTIFIER MODIFICATION DOSSIER RESOURCES

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande NOTIFIER MODIFICATION DOSSIER RESOURCES permet de "forcer" l'envoi par 4D Server d'une notification indiquant à tous les postes 4D connectés que le contenu du dossier Resources de la base a été modifié, afin de leur permettre de synchroniser leur dossier Resources local. Cette commande permet en particulier de gérer la synchronisation des dossiers Resources téléchargés sur les postes distants lorsque le dossier Resources de la base est modifié via une procédure stockée sur le serveur.
Pour plus d'informations sur la gestion du dossier Resources en mode distant, reportez-vous au*Guide de référence* de 4D Server.

Seule l'information de modification est envoyée par cette commande. Chaque poste distant réagira en fonction de ses préférences, définies en local. Les options sont :
• pas de synchronisation du dossier Resources local en cours de session,
• synchronisation automatique du dossier Resources local en cours de session,
• affichage d'une alerte afin que l'utilisateur effectue une synchronisation s'il le souhaite.

Référence

Dossier 4D.

OUVRIR CENTRE DE SECURITE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande OUVRIR CENTRE DE SECURITE provoque l'affichage de la fenêtre du Centre de sécurité et de maintenance (CSM).

En fonction des privilèges d'accès de l'utilisateur courant, certaines fonctions proposées dans la fenêtre pourront être désactivées.

Référence

VERIFIER FICHIER DONNEES OUVERT.

OUVRIR FENETRE ADMINISTRATION

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande OUVRIR FENETRE ADMINISTRATION affiche la fenêtre d'administration du serveur sur le poste client 4D distant qui l'exécute. La fenêtre d'administration de 4D Server permet de visualiser les paramètres courants et d'effectuer diverses opérations de maintenance (cf. *Guide de référence* de 4D Server). A compter de la version 11 de 4D Server, cette fenêtre peut être affichée depuis un poste client :

Cette commande doit être appelée dans le contexte d'une application 4D connectée à un 4D Server. Elle ne fait rien si :

• elle est appelée dans une application 4D en mode local ou exécutée en procédure stockée sur le serveur,

• elle est exécutée par utilisateur autre que le Super_Utilisateur ou l'Administrateur (dans ce cas, l'erreur -9991 est générée, cf. section Erreurs de la base de données).

Exemples

Voici le code d'un bouton d'administration :

```
Si (Type application=4D mode local)
   OUVRIR CENTRE DE SECURITE
   ` ...
Fin de si
Si (Type application=4D mode distant)
   OUVRIR FENETRE ADMINISTRATION
   ` ...
Fin de si
Si (Type application=4D Server)
   ` ...
Fin de si
```

Référence

OUVRIR CENTRE DE SECURITE.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système *OK* prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

OUVRIR FICHIER DONNEES (cheminAccès)

Paramètre	Type		Description
cheminAccès	Alpha	→	Nom ou chemin d'accès complet du fichier de données à ouvrir

Description

La commande OUVRIR FICHIER DONNEES permet de changer à la volée le fichier de données ouvert par l'application 4D.

Vous passez dans le paramètre cheminAccès le nom ou le chemin d'accès complet du fichier de données à ouvrir. Si vous passez uniquement un nom de fichier, il doit être placé à côté du fichier de structure de la base.

Si ce chemin d'accès désigne un fichier de données valide, 4D quitte la base en cours et la rouvre avec le fichier de données spécifié. La Méthode base Sur fermeture et la Méthode base Sur ouverture sont successivement appelées.

Attention : Comme cette commande provoque la fermeture préalable de l'application, il n'est pas possible de l'utiliser dans la Méthode base Sur ouverture ou une méthode appelée par cette méthode base.

La commande est exécutée de manière asynchrone : après son appel, 4D continue d'exécuter le reste de la méthode. Ensuite, l'application se comporte comme si la commande **Quitter** avait été sélectionnée dans le menu **Fichier** : les boîtes de dialogue ouvertes sont annulées, les process ouverts ont 10 secondes pour se terminer avant d'être tués, etc.

Avant de lancer l'opération, la commande teste la validité du fichier de données spécifié. En outre, si le fichier a déjà été ouvert, la commande vérifie qu'il correspond bien à la structure courante.

Si vous passez une chaîne vide dans cheminAccès, la commande rouvre la base sans changer de fichier de données.

4D Server : Cette commande ne peut pas être utilisée avec 4D Client ou 4D Server.

Référence

CREER FICHIER DONNEES.

OUVRIR PREFERENCES 4D (sélecteur)

Paramètre	Type		Description
sélecteur	Chaîne	→	Clé désignant un thème ou une page ou un groupe de paramètres de la boîte de dialogue des Préférences

Description

La commande OUVRIR PREFERENCES 4D provoque l'ouverture de la boîte de dialogue des Préférences de l'application 4D courante et l'affichage du thème ou de la page correspondant à la clé passée dans le paramètre sélecteur.

Le paramètre sélecteur doit contenir une ou plusieurs "clés" indiquant un thème, une page ou un groupe de paramètres de la boîte de dialogue des Préférences. La liste des clés utilisables est fournie ci-dessous.

Vous pouvez passer dans sélecteur soit un "chemin d'accès" absolu, soit un nom d'élément simple :
• **chemin d'accès absolu** : le paramètre sélecteur est construit de la manière suivante : /Thème{/Page{/Groupe de paramètre}}.
La chaîne doit débuter par le caractère / et chaque niveau doit être séparé par un /.
Par exemple, pour désigner la page Compilateur du thème Mode Développement, sélecteur doit contenir "/Design Mode/Compiler".
• **nom** (chemin relatif) : dans ce cas, le paramètre sélecteur ne doit pas débuter par le caractère /. Il suffit de passer le nom de l'élément souhaité et 4D ouvrira le premier élément correspondant dans l'ordre de recherche groupe de paramètres -> page -> thème.
Par exemple, si vous passez "Progress Indicator" dans sélecteur, 4D ouvrira la page Options du thème Application.

Pour ouvrir directement la boîte de dialogue sur la première page, passez simplement "/" dans sélecteur.

La commande ouvre la page des Préférences sur l'élément spécifié dans sélecteur, mais tous les autres thèmes et pages restent accessibles. Il est du ressort du développeur de s'assurer que l'accès des utilisateurs aux Préférences ne risque pas de perturber le fonctionnement de l'application. Pour contrôler les actions réalisables par les utilisateurs, il est recommandé d'activer le système de gestion des droits d'accès.

Clés de chemins

Voici la liste des clés utilisables dans le paramètre sélecteur :

/Application
/Application/Options
/Application/Options/Options
/Application/Options/Temporary Folder Location
/Application/Options/Drag and Drop Highlight
/Application/Options/Progress Indicator
/Application/Options/Display Toolbar
/Application/Options/Display Windows
/Application/Access
/Application/Access/Data Access
/Application/Access/General Settings
/Application/Access/User Access
/Application/CPU Priorities
/Application/CPU Priorities/Set CPU Priority to:
/Application/Shortcuts
/Application/Shortcuts/Keys
/Application/Compatibility
/Application/Compatibility/Design Compatibility
/Application/Compatibility/Web Compatibility
/Application/Compatibility/Platform
/Design Mode
/Design Mode/Structure
/Design Mode/Structure/General Font
/Design Mode/Structure/Forms and Methods Automatic Comments
/Design Mode/Structure/Automatic Form Creation
/Design Mode/Form Editor
/Design Mode/Form Editor/Object Templates
/Design Mode/Form Editor/Move
/Design Mode/Form Editor/Auto Alignment
/Design Mode/Form Editor/Default Display
/Design Mode/Method Editor
/Design Mode/Method Editor/Font
/Design Mode/Method Editor/Default Display
/Design Mode/Method Editor/Options
/Design Mode/Method Editor/Syntax
/Design Mode/Compiler
/Design Mode/Compiler/Compilation Options
/Design Mode/Compiler/Compiler Methods for...
/Design Mode/Documentation
/Design Mode/Moving
/Design Mode/Moving/Default Actions during the Copy if Dependent Objects
/Design Mode/Moving/Moving Dialog

/Database
/Database/Data Management
/Database/Data Management/General
/Database/Data Management/Database Cache Settings
/Database/Data Management/WEDD
/Database/International
/Database/International/Script Manager
/Database/International/Right-to-left Languages
/Database/International/Numeric Display Format
/Backup
/Backup/Configuration
/Backup/Configuration/Backup Contents
/Backup/Configuration/Backup File Destination Folder
/Backup/Configuration/Last Backup Information
/Backup/Configuration/Log Management
/Backup/Scheduler
/Backup/Scheduler/Backup Frequency
/Backup/Backup
/Backup/Backup/General
/Backup/Backup/Archive
/Backup/Restore
/Backup/Restore/Automatic Restore
/Client-Server
/Client-Server/Configuration
/Client-Server/Configuration/Network
/Client-Server/Configuration/Client-Server Connections Timeout
/Client-Server/Configuration/Client-Server Communication
/Client-Server/Configuration/4D Open
/Client-Server/Publishing
/Client-Server/Publishing/Publishing Information
/Client-Server/Publishing/Allow-Deny Table Configuration
/Client-Server/Publishing/Encryption
/Web
/Web/Configuration
/Web/Configuration/Web Server Publishing
/Web/Configuration/Default HTML Path
/Web/Configuration/Starting Mode
/Web/Advanced
/Web/Advanced/Cache
/Web/Advanced/Web Process
/Web/Advanced/Options
/Web/Advanced/Web Passwords
/Web/Options
/Web/Options/Options
/Web/Options/Text Conversion

/Web/Options/Persisent Connections
/Web/Log Format
/Web/Log Format/Web Log Type
/Web/Log Format/Web Log Token Selection
/Web/Log Scheduler
/Web/Log Scheduler/Backup Frequency for Web Log File
/Web Services
/Web Services/SOAP
/Web Services/SOAP/Server Side
/Web Services/SOAP/Client Side
/SQL
/SQL/Configuration
/SQL/Configuration/SQL Server Access

Exemples

(1) Ouverture des Préférences sur la première page :

```
OUVRIR PREFERENCES 4D("/")
```

(2) Ouverture de la page “Raccourcis clavier” du thème “Application” :

```
OUVRIR PREFERENCES 4D("/Application/Shortcuts")
```

(3) Ouverture de la page “Avancé” du thème “Web” :

```
OUVRIR PREFERENCES 4D("Web Passwords")
```

Variables et ensembles système

Si l’élément demandé est trouvé et correctement ouvert, la variable système *OK* retourne 1 ; sinon, elle retourne 0.

QUITTER 4D {(délai)}

Paramètre	Type		Description
délai	Numérique	→	Délai (mn) avant que le serveur ne quitte

Description

La commande QUITTER 4D vous permet de quitter l'application 4D courante et de retourner sur le Bureau du système d'exploitation.

Le mécanismes mis en jeu par la commande sont différents suivant qu'elle est exécutée sur 4D ou 4D Server (procédure stockée).

Avec 4D en mode local ou distant

Après un appel à QUITTER 4D, l'exécution du process courant est stoppée, puis 4D effectue les opérations suivantes :

• Si une Méthode base Sur fermeture existe, 4D l'exécute dans un nouveau process local. Par exemple, vous pouvez utiliser cette méthode base pour informer les autres process, via la communication interprocess, qu'ils doivent être fermés (s'ils sont en saisie de données) ou stopper l'exécution des opérations démarrées dans la Méthode base Sur ouverture (connexion de 4D à un autre serveur de bases de données). Notez que 4D quittera dans tous les cas : la Méthode base Sur fermeture peut assurer le nettoyage et la fermeture de toutes les opérations que vous voulez, mais la fermeture de la base est inéluctable.

• S'il n'existe pas de Méthode base Sur fermeture, 4D ferme tous les process un par un, sans distinction.

Si l'utilisateur est en saisie de données, les enregistrements seront annulés et non validés.
Si vous voulez permettre à l'utilisateur de sauvegarder ses modifications effectuées dans les fenêtres du process courant, vous pouvez utiliser la communication interprocess pour indiquer à tous les autres process utilisateur que la base est sur le point d'être quittée. Pour cela, vous pouvez adopter deux stratégies :

• Effectuer ces opérations depuis le process courant avant d'appeler QUITTER 4D.
• Traiter ces opérations depuis la Méthode base Sur fermeture.

Une troisième stratégie est également possible. Avant d'appeler QUITTER 4D, vous testez si une fenêtre nécessite une validation. Si c'est le cas, vous demandez à l'utilisateur de valider ou d'annuler cette fenêtre puis de choisir Quitter de nouveau. Cependant, du point de vue purement "interface utilisateur", les deux premières solutions sont préférables.

Note : Le paramètre délai n'est pas utilisable avec 4D.

Avec 4D Server (procédure stockée)
La commande QUITTER 4D peut être exécutée sur le poste serveur, dans une procédure stockée. Dans ce cas, elle admet le paramètre optionnel délai. Ce paramètre permet d'allouer à 4D Server un délai d'attente avant que l'application ne quitte réellement, laissant ainsi aux postes clients le temps de se déconnecter. Vous devez passer dans délai une valeur en minutes.
Ce paramètre n'est pris en compte que dans le cadre d'une exécution sur le poste serveur. Avec 4D, il est ignoré.
Si vous ne passez pas le paramètre délai, 4D Server attendra que tous les postes clients soient déconnectés avant de quitter.
A la différence de 4D, le traitement de QUITTER 4D par 4D Server est asynchrone : la méthode dans laquelle la commande est appelée n'est pas interrompue après son exécution.

Si une Méthode base Sur arrêt serveur existe, elle est exécutée à l'issue du délai défini par le paramètre délai, ou de la déconnexion de tous les clients, suivant vos paramétrages.

L'action de la commande QUITTER 4D utilisée dans une procédure stockée est équivalente à celle de la commande **Quitter** du menu **Fichier** de 4D Server : elle provoque l'apparition, sur chaque poste client, d'une boîte de dialogue signalant que le serveur est sur le point de quitter.

Exemple
La méthode projet suivante est associée à la commande **Quitter** du menu **Fichier**.

```
      ` Méthode projet M_QUITTER

CONFIRMER("Etes-vous certain de vouloir quitter ?")
Si (OK=1)
   QUITTER 4D
Fin de si
```

Référence
Méthode base Sur arrêt serveur, Méthode base Sur fermeture.

Type application

Type application → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Valeur numérique représentant le type de l'application

Description

La fonction Type application renvoie une valeur numérique qui représente le type de l'environnement 4D que vous utilisez. 4D vous fournit les constantes prédéfinies suivantes, placées dans le thème “Environnement 4D” :

Constante	Type	Valeur
4D mode local	Entier long	0
4D Volume Desktop	Entier long	1
4D mode distant	Entier long	4
4D Server	Entier long	5
4D First	Entier long	6

Exemple

Quelque part dans votre code, ailleurs que dans la Méthode base Sur démarrage serveur, vous voulez vérifier si l'utilisateur a ouvert la base avec 4D Server. Pour cela, vous pouvez écrire les lignes de code suivantes :

```
Si (Type application=4D Server)
   ` Exécuter des actions nécessaires
Fin de si
```

Référence

Type version, Version application.

Type version → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	0 = Version standard 1 = Version de démonstration

Description

La commande Type version retourne une valeur numérique qui représente le type de version de 4D que vous utilisez. 4D vous fournit les constantes prédéfinies suivantes :

Constante	Type	Valeur
Version standard	Entier long	0
Version de démonstration	Entier long	1

Exemple

Votre application 4D inclut des fonctionnalités qui ne sont pas disponibles lorsque vous êtes en version démo. Vous pouvez tester l'environnement en écrivant le code suivant :

```
Si (Type version=Version standard)
   ` Exécuter les actions nécessaires
Sinon
   ALERTE("Cette fonctionnalité n'est pas disponible dans la version démo de"+
                                                    "Super Gestion™.")
Fin de si
```

Référence

Type application, Version application.

VERIFIER FICHIER DONNEES (cheminStructure; cheminDonnées; objets; options; méthode{; tabTables; tabChamps})

Paramètre	Type		Description
cheminStructure	Texte	→	Chemin d'accès du fichier de structure de la base à vérifier
cheminDonnées	Texte	→	Chemin d'accès du fichier de données de la base à vérifier
objets	Num	→	Objets à vérifier
options	Num	→	Options de vérification
méthode	Texte	→	Nom de la méthode 4D de rétroappel
tabTables	Tab num	→	Numéros des tables à vérifier
tabChamps	Tab num 2D	→	Numéros des index à vérifier

Description

La commande VERIFIER FICHIER DONNEES effectue une vérification structurelle des objets contenus dans le fichier de données 4D désigné par cheminStructure et cheminDonnées.

Note : Pour plus d'informations sur le processus de vérification des données, reportez-vous au manuel *Mode Développement*.

• cheminStructure désigne le fichier de structure (compilé ou non) associé au fichier de données à vérifier. Il peut s'agir du fichier de structure ouvert ou de tout autre fichier de structure. Vous devez passer un chemin d'accès complet, exprimé avec la syntaxe du système d'exploitation. Vous pouvez également passer une chaîne vide, dans ce cas une boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de désigner le fichier de structure à utiliser.

• cheminDonnées désigne un fichier de données 4D (.4DD). Il doit correspondre au fichier de structure défini par le paramètre cheminStructure. Attention, vous pouvez désigner le fichier de structure courant mais le fichier de données ne doit pas être le fichier courant (ouvert). Pour vérifier le fichier de données ouvert, utilisez la commande VERIFIER FICHIER DONNEES OUVERT. Si vous tentez de vérifier le fichier de données courant avec la commande VERIFIER FICHIER DONNEES, une erreur est générée.
Le fichier de données désigné est ouvert en lecture seulement. Vous devez veiller à ce qu'aucune application n'accède à ce fichier en écriture, sinon les résultats de la vérification pourront être faussés.

Vous pouvez passer dans le paramètre cheminDonnées une chaîne vide, un nom de fichier ou un chemin d'accès complet, exprimé dans la syntaxe du système d'exploitation. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichier apparaît, permettant à l'utilisateur de désigner le fichier à vérifier (à noter dans ce cas qu'il n'est pas possible de sélectionner le fichier de données courant). Si vous passez uniquement un nom de fichier de données, 4D le recherchera au même niveau que le fichier de structure défini.

• Le paramètre objets permet de désigner le(s) type(s) d'objets à vérifier. Deux types d'objets peuvent être vérifiés : les enregistrements et les index. Utilisez les constantes suivantes, placés dans le thème "Maintenance fichier de données" :
- Vérifier enregistrements (4)
- Vérifier index (8)
- Tout vérifier (16)

Pour vérifier les enregistrements et les index, passez le cumul Vérifier enregistrements+Vérifier index. La valeur 0 (zéro) permet également d'obtenir le même résultat. L'option Tout vérifier effectue la vérification interne la plus complète. Cette vérification est compatible avec la création d'un historique.

• Le paramètre options permet de définir les options de vérification. Une seule option est actuellement disponible, accessible via une constante du thème "Maintenance fichier de données" : Ne pas créer d'historique (16384).
En principe, la commande VERIFIER FICHIER DONNEES crée un fichier d'historique au format xml (reportez-vous à la fin de la description de cette commande). Vous pouvez annuler ce fonctionnement en passant cette option. Pour créer l'historique, passez 0 dans options.

• Le paramètre méthode permet de définir une méthode de rétro-appel qui sera régulièrement appelée durant la vérification. Si vous passez une chaîne vide, aucune méthode n'est appelée. Si la méthode passée n'existe pas, la vérification n'est pas effectuée, une erreur est générée et la variable OK prend la valeur 0. Lorsqu'elle est appelée, cette méthode reçoit jusqu'à 5 paramètres en fonction du type d'événement à l'origine de l'appel (cf. tableau des appels). Vous devez impérativement déclarer ces paramètres dans la méthode :

- $1	Entier long	Type de message (cf. tableau)
- $2	Entier long	Type d'objet
- $3	Texte	Message
- $4	Entier long	Numéro de table
- $5	Entier long	Réservé

Le tableau suivant décrit le contenu des paramètres en fonction du type d'événement :

Evénement	$1 (E. long)	$2 (E. long)	$3 (Texte)	$4 (E. long)	$5 (E. long)
Message	1	0	Message de progression	Pourcentage réalisé (0-100)	Réservé
Vérification OK	2	Type d'objet	Texte du message OK	Numéro de table ou d'index	Réservé
Erreur	3	Type d'objet	Texte du message d'erreur	Numéro de table ou d'index	Réservé
Fin d'exécution	4	0	DONE	0	Réservé
Warning	5	Type d'objet	Texte du message d'erreur	Numéro de table ou d'index	Réservé

Type d'objet : Lorsqu'un objet a été vérifié, un message OK ($1=2), erreur ($1=3) ou warning ($1=5) peut être envoyé. Le type d'objet retourné dans $2 peut être l'un des suivants :
• 0 = indéterminé
• 4 = enregistrement
• 8 = index
• 16 = objet structure (contrôle préliminaire du fichier de données).

Cas particulier : lorsque $4 = 0 pour $1 = 2, 3 ou 5, le message ne concerne pas une table mais le fichier de données dans son ensemble.

La méthode de rétro-appel doit également retourner une valeur dans $0 (Entier long), permettant de contrôler l'exécution de l'opération :
- si $0 = 0, l'opération continue normalement
- si $0 = -128, l'opération est stoppée sans erreur générée
- si $0 = autre valeur, l'opération est stoppée et la valeur passée dans $0 est retournée en tant que numéro d'erreur. Cette erreur peut être interceptée par une méthode d'appel sur erreur.

Deux tableaux facultatifs peuvent également être utilisés par la commande :
• Le tableau tabTables contient les numéros des tables dont les enregistrements doivent être vérifiés. Il permet de limiter la vérification à certaines tables. Si ce paramètre n'est pas passé ou si le tableau est vide et que le paramètre objets contient Vérifier enregistrements, toutes les tables sont vérifiées.
• Le tableau tabChamps contient les numéros des champs indexés dont les index doivent être vérifiés. Si ce paramètre n'est pas passé ou si le tableau est vide et que le paramètre objets contient Vérifier index, tous les index sont vérifiés. La commande ignore les champs non indexés. Si un champ contient plusieurs index, tous les index sont vérifiés. Si un champ fait partie d'un index composite, la totalité de l'index est vérifiée.
Vous devez passer un tableau 2D dans tabChamps. Pour chaque ligne du tableau :
- l'élément {0} contient le numéro de la table,
- les autres éléments {1...n} contiennent les numéros des champs.

Par défaut, la commande VERIFIER FICHIER DONNEES crée un fichier d'historique au format xml (si vous n'avez pas passé l'option Ne pas créer d'historique, cf. paramètre options).

Son nom est basé sur celui du fichier de données et il est placé dans le dossier "Logs" de la base. Par exemple, pour un fichier de données nommé “data.4dd”, le fichier d’historique sera nommé “data_verify_log.xml”.

Exemples

(1) Vérification simple des données et des index :

```
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier index+
                                    Vérifier enregistrements;Ne pas créer d’historique;"")
```

(2) Vérification complète avec historique :

```
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Tout vérifier;0;"")
```

(3) Vérification des enregistrements uniquement :

```
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier enregistrements;0;"")
```

(4) Vérification des enregistrements des tables 3 et 7 uniquement :

```
TABLEAU ENTIER LONG($tnumTables;2)
TABLEAU ENTIER LONG($tindex;0) `non utilisé mais obligatoire
$tnumTables{1}:=3
$tnumTables{2}:=7
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier enregistrements;
                                    0;"FollowScan";$tnumTables;$tindex)
```

(5) Vérification d’index spécifiques (index du champ 1 de la table 4 et index des champs 2 et 3 de la table 5) :

```
TABLEAU ENTIER LONG($tnumTables;0) `non utilisé mais obligatoire
TABLEAU ENTIER LONG($tindex;2;0) `2 lignes (colonnes ajoutées ensuite)
$tindex{1}{0}:=4 ` numéro de table dans l’élément 0
AJOUTER A TABLEAU($tindex{1};1) ` numéro du 1er champ à vérifier
$tindex{2}{0}:=5 ` numéro de table dans l’élément 0
AJOUTER A TABLEAU($tindex{2};2) ` numéro du 1er champ à vérifier
AJOUTER A TABLEAU($tindex{2};3) ` numéro du 2e champ à vérifier
VERIFIER FICHIER DONNEES($NomStruct;$NomData;Vérifier index;0;"FollowScan";
                                    $tnumTables;$tindex)
```

Référence

VERIFIER FICHIER DONNEES OUVERT.

Variables et ensembles système

Si la méthode de rétro-appel passée n’existe pas, la vérification n’est pas effectuée, une erreur est générée et la variable *OK* prend la valeur 0.

VERIFIER FICHIER DONNEES OUVERT{ (objets; options; méthode{; tabTables; tableChamps})}

Paramètre	Type		Description
objets	Numérique	→	Objets à vérifier
options	Numérique	→	Options de vérification
méthode	Texte	→	Nom de la méthode 4D de rétro-appel
tabTables	Tab Num	→	Numéros des tables à vérifier
tableChamps	Tab Num 2D	→	Numéros des index à vérifier

Description

La commande VERIFIER FICHIER DONNEES OUVERT effectue une vérification structurelle des objets contenus dans le fichier de données actuellement ouvert par 4D.

Cette commande a un fonctionnement identique à celui de la commande VERIFIER FICHIER DONNEES, à la différence près qu'elle s'applique uniquement au fichier de données courant de la base de données ouverte. Elle ne nécessite donc pas de paramètres désignant la structure et les données.
Reportez-vous à la commande VERIFIER FICHIER DONNEES pour la description des paramètres.

Si vous passez directement la commande VERIFIER FICHIER DONNEES OUVERT sans aucun paramètre, la vérification est effectuée avec les valeurs par défaut des paramètres :
- objets = Tout vérifier (= valeur 16)
- options = 0 (l'historique est créé)
- méthode = ""
- tabTables et tabChamps sont omis.

Lorsque cette commande est exécutée, le cache de données est écrit sur le disque et toutes les opérations accédant aux données sont bloquées durant la vérification.

Note : Cette commande ne doit pas être utilisée lorsqu'un process est en cours de création d'index ou de mise à jour d'index car dans ce cas, les résultats de la vérification seront invalides.

Référence

VERIFIER FICHIER DONNEES.

Version application {(*)} → Alpha

Paramètre	Type		Description
*	*	→	Si passé = numéro de version long Si omis = numéro de version court
Résultat	Alpha	←	Numéro de version dans une chaîne encodée

Description

Version application retourne une chaîne encodée qui exprime le numéro de version de l'environnement 4D que vous utilisez.

• Si vous ne passez pas le paramètre optionnel *, une chaîne de 4 caractères est retournée, formatée de la manière suivante :

Caractères	Description
1-2	Numéro de version
3	Numéro de mise à jour
4	Numéro de révision

Exemple : la chaîne "0600" représente la version 6.0.0.

• Si vous passez le paramètre optionnel *, une chaîne de 8 caractères est retournée, formatée de la manière suivante :

Caractères	Description
1	"F" représente une version finale "B" représente une version beta Les autres caractères représentent une version interne à 4D
2-3-4	Numéro de compilation interne à 4D
5-6	Numéro de version
7	Numéro de mise à jour
8	Numéro de révision

Exemple : la chaîne "B0120602" représente une version beta 12 de la version 6.0.2.

Exemples

(1) Cet exemple affiche le numéro de version de l'environnement 4D :

```
$vs4Dversion:=Version application
ALERTE("Vous utilisez la version "+Chaine(Num(Sous chaine($vs4Dversion;1;2)))+"."+
                                                    $vs4Dversion[[3]]+"."+$vs4Dversion[[4]])
```

(2) Cet exemple teste si vous utilisez une version finale :

```
Si(Sous chaine(Version application(*);1;1)#"F")
    ALERTE("Veuillez vous assurer que vous utilisez une version finale de 4D avec cette
                                                                                base !")
    QUITTER 4D
Fin de si
```

Référence

Type application, Type version.

17

Environnement système

COORDONNEES ECRAN (gauche; haut; droite; bas{; écran})

Paramètre	Type		Description
gauche	Entier long	←	Coordonnée gauche de la zone de l'écran
haut	Entier long	←	Coordonnée supérieure de la zone de l'écran
droite	Entier long	←	Coordonnée droite de la zone de l'écran
bas	Entier long	←	Coordonnée inférieure de la zone de l'écran
écran	Entier long	→	Numéro de l'écran ou écran principal si omis

Description

La commande COORDONNEES ECRAN retourne dans les paramètres gauche, haut, droite et bas les coordonnées de l'écran spécifié dans le paramètre écran.

Si vous omettez le paramètre écran, COORDONNEES ECRAN retourne les coordonnées de l'écran principal.

Référence

Ecran principal, Nombre ecrans, PROFONDEUR ECRAN.

Dossier systeme {(type)} → Alpha

Paramètre	Type		Description
type	Entier long	→	Type de dossier système
Résultat	Alpha	←	Chemin d'accès d'un dossier du système actif

Description

La fonction Dossier systeme retourne le chemin d'accès du dossier Système Windows ou Mac OS actif, ou le chemin d'accès d'un dossier particulier du système d'exploitation.

Le paramètre optionnel type vous permet d'indiquer le type de dossier dont vous souhaitez obtenir le chemin d'accès. Si vous ne passez pas ce paramètre, Dossier systeme retourne le chemin d'accès du dossier Système Windows ou Mac OS actif.
Vous passez dans type un code représentant le type de dossier. 4D fournit les constantes prédéfinies suivantes (placées dans le thème "Dossier système") :

Constante	Type	Valeur
Système	Entier long	0
Polices	Entier long	1
Préférences utilisateur_Tous	Entier long	2
Préférences utilisateur	Entier long	3
Démarrage Win_Tous	Entier long	4
Démarrage Win	Entier long	5
Menu Démarrer Win_Tous	Entier long	8
Menu Démarrer Win	Entier long	9
System Win	Entier long	12
System32 Win	Entier long	13
Favoris Win	Entier long	14
Bureau	Entier long	15
Applications ou Program Files	Entier long	16

Notes :

• Les constantes suffixées Win sont réservées à une utilisation sous Windows. Lorsqu'elles sont utilisées sous Mac OS, Dossier systeme retourne une chaîne vide.

• L'emplacement de certains dossiers peut être différent suivant le type de session ouverte par l'utilisateur. Les constantes 2 à 9 permettent de choisir si vous souhaitez obtenir le chemin d'accès du dossier spécifique à l'utilisateur courant (constantes simples) ou commun à tous les utilisateurs (constantes suivies de "Tous").

Référence

Dossier 4D, Dossier temporaire.

Dossier temporaire → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Chemin d'accès au dossier temporaire

Description

La fonction Dossier temporaire retourne le chemin d'accès au dossier temporaire courant tel que défini par votre système d'exploitation.

Exemple

Reportez-vous à l'exemple de la commande AJOUTER DONNEES AU CONTENEUR.

Référence

Dossier systeme.

Ecran principal → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Numéro de l'écran contenant la barre de menus

Description

La commande Ecran principal retourne le numéro de l'écran dans lequel se trouve la barre de menus.

Note pour les utilisateurs Windows : Sous Windows, Ecran principal renvoie toujours 1.

Référence

Hauteur barre de menus, Nombre ecrans.

ENREGISTRER EVENEMENT ({typeSortie; }message{; importance})

Paramètre	Type		Description
typeSortie	Entier	→	Type de sortie du message
message	Alpha	→	Contenu du message
importance	Entier	→	Niveau d'importance du message (Windows uniquement)

Description

La commande ENREGISTRER EVENEMENT vous permet de mettre en place un système personnalisé d'enregistrement des événements internes qui se produisent au cours de l'utilisation de votre application. Vous pouvez ainsi contrôler le déroulement d'une session de travail

Passez dans le paramètre message les informations personnalisées à noter en fonction de l'événement.

Le paramètre facultatif typeSortie vous permet de préciser le canal de sortie emprunté par le message. Vous pouvez passer dans ce paramètre une des constantes suivantes, placées dans le thème "Journal d'événements" :

Constante	Type	Valeur
Vers Observateur Windows	Entier long	0
Vers message débogage	Entier long	1
Vers historique requêtes 4D	Entier long	2
Vers historique commandes 4D	Entier long	3

Note : Si vous omettez le paramètre typeSortie, la commande utilise le canal 0 (Vers Observateur Windows).

• Vers Observateur Windows : cette valeur indique à 4D d'envoyer le message vers l'"Observateur d'événements" de Windows. Ce journal reçoit et stocke des messages en provenance des applications en cours d'exécution.

Notes :
- Pour que cette fonctionnalité soit disponible, le service Observateur d'événements de Windows doit être démarré.
- Sous Mac OS, la commande ne fait rien avec ce type de sortie.

• Vers message débogage : cette valeur indique à 4D d'envoyer le message dans l'environnement de débogage du système. Le résultat dépend de la plate-forme :
- sous Mac OS : la commande envoie le message à la Console
- sous Windows : la commande envoie le message en tant que message de débogage. Pour pouvoir lire ce message, vous devez disposer de Microsoft Visual Studio ou de l'utilitaire DebugView pour Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx).

• Vers historique requêtes 4D : cette valeur indique à 4D d'inscrire le message dans le fichier d'historique des requêtes de 4D, si ce fichier a été activé.

• Vers historique commandes 4D : cette valeur indique à 4D d'inscrire le message dans le fichier d'historique des commandes de 4D, si ce fichier a été activé.

Note : Les fichiers d'historique de 4D sont regroupés dans le dossier **Logs**, créé à côté du fichier de structure de la base (cf. commande Dossier 4D).

Si vous avez défini un typeSortie de type Vers Observateur Windows, vous pouvez attribuer au message un niveau d'importance via le paramètre facultatif importance afin de faciliter la lecture du journal d'événements. Il existe trois niveaux d'importance : Information, Avertissement et Erreur. 4D vous propose les constantes prédéfinies suivantes, placées dans le thème "Journal d'événements" :

Constante	Type	Valeur
Message d'information (valeur par défaut)	Entier long	0
Message d'avertissement	Entier long	1
Message d'erreur	Entier long	2

Si vous ne passez pas le paramètre importance ou passez une valeur invalide, la valeur par défaut (0) est utilisée.

Exemple

Afin de conserver une trace des lancements de votre base sous Windows, vous pouvez écrire, dans la Méthode base Sur ouverture :

```
ENREGISTRER EVENEMENT (Vers Observateur Windows;"Démarrage de la base
                                                    Facturation")
```

A chaque ouverture de la base, cette information sera inscrite dans l'Observateur d'événements de Windows, avec le niveau d'importance 0.

FIXER PROFONDEUR ECRAN (profondeur{; couleurs{; écran}})

Paramètre	Type		Description
profondeur	Numérique	→	Profondeur de l'écran (nombre de couleurs = 2 ^ profondeur)
couleurs	Numérique	→	1 = écran couleur 0 = écran en niveaux de gris
écran	Numérique	→	Numéro de l'écran ou écran principal si omis

Description

FIXER PROFONDEUR ECRAN vous permet de modifier la profondeur et les paramètres de couleurs/niveaux de gris de l'écran dont vous avez passé le numéro dans écran. Si vous ne passez pas ce paramètre, la commande s'applique à l'écran principal.

Pour connaître les valeurs à passer dans les paramètres profondeur et couleurs, reportez-vous à la description de la commande PROFONDEUR ECRAN.

Référence

PROFONDEUR ECRAN.

Gestalt (sélecteur; valeur) → Numérique

Paramètre	Type		Description
sélecteur	Alpha	→	Sélecteur gestalt (4 caractères)
valeur	Numérique	←	Résultat du gestalt
Résultat	Numérique	←	Code d'erreur résultant

Description

La commande Gestalt retourne dans valeur une valeur numérique représentant les caractéristiques de la configuration matérielle et logicielle de votre système, en fonction du sélecteur que vous avez passé dans le paramètre sélecteur.

Si l'information demandée est obtenue, la fonction Gestalt retourne 0, sinon elle retourne l'erreur -5550. Si le sélecteur est inconnu, Gestalt retourne l'erreur -5551.

Important : Le Gestalt Manager est spécifique à Mac OS. Certains des sélecteurs sont également implémentés sous Windows mais l'utilité de cette fonction reste limitée sur cette plate-forme.

Pour plus d'informations sur les sélecteurs que vous pouvez passer dans Gestalt, reportez-vous à la documentation technique Apple relative au Gestalt Manager, consultable en ligne à l'adresse suivante :
http://developer.apple.com/documentation/Carbon/Reference/Gestalt_Manager/index.html

Exemple

Dans la version 10.4.11 de Mac OS, le code suivant :

```
$vlErrCode:=Gestalt("sysv";$vlInfo)
Si ($vlErrCode=0)
    ALERTE("Vous utilisez la version suivante du système : "+Chaîne($vlInfo;"&x"))
Fin de si
```

... affiche l'alerte “Vous utilisez la version suivante du système : 0x1049”.

Hauteur barre de menus

Hauteur barre de menus → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Hauteur (exprimée en pixels) de la barre de menus (retourne zéro si la barre de menus est cachée)

Description

La commande Hauteur barre de menus retourne la hauteur de la barre de menus, exprimée en pixels.

Référence

AFFICHER BARRE DE MENUS, CACHER BARRE DE MENUS, Ecran principal.

Hauteur ecran {(*)} → Numérique

Paramètre	Type		Description
*	*	→	Windows : hauteur de la fenêtre de l'application ou hauteur de l'écran si * est spécifié Macintosh : hauteur de l'écran principal
Résultat	Numérique	←	Hauteur exprimée en pixels

Description

Sous Windows, Hauteur ecran retourne la hauteur de la fenêtre d'application 4D (fenêtre MDI). Si vous passez le paramètre optionnel *, Hauteur ecran retourne la hauteur de l'écran.

Sous Mac OS, Hauteur ecran retourne la hauteur de l'écran principal, c'est-à-dire celui qui contient la barre de menus.

Référence

COORDONNEES ECRAN, Largeur ecran.

Largeur ecran {(*)} → Numérique

Paramètre	Type		Description
*	*	→	Windows : largeur de la fenêtre de l'application ou largeur de l'écran si * est spécifié Macintosh : largeur de l'écran principal
Résultat	Numérique	←	Largeur exprimée en pixels

Description

Sous Windows, Largeur ecran retourne la largeur de la fenêtre d'application 4D (fenêtre MDI). Si vous passez le paramètre optionnel *, Largeur ecran retourne la largeur de l'écran.

Sous Mac OS, Largeur ecran retourne la largeur de l'écran principal, c'est-à-dire celui qui contient la barre de menus.

Référence

COORDONNEES ECRAN, Hauteur ecran.

LIRE FORMATAGE SYSTEME (formatage; valeur)

Paramètre	Type		Description
formatage	Entier long	→	Formatage système à lire
valeur	Chaîne	←	Valeur de formatage définie dans le système

Description

La commande LIRE FORMATAGE SYSTEME retourne la valeur courante de plusieurs paramètres régionaux définis dans le système d'exploitation. Cette commande permet de construire des formats personnalisés "automatiques" basés sur les préférences système.

Passez dans le paramètre formatage le type de paramètre dont vous souhaitez connaître la valeur. Le résultat est retourné directement par le système dans le paramètre valeur sous forme de chaîne de caractères. Vous devez passer dans formatage une des constantes du thème "Formatages système". Voici le descriptif de ces constantes :

Constante (valeur)	Valeur(s) retournée(s)
Séparateur décimal (0)	Séparateur décimal (ex : ",")
Séparateur de milliers (1)	Séparateur de milliers (ex : " ")
Symbole monétaire (2)	Symbole monétaire (ex : "$")
Motif heure court (3)	Format d'affichage d'heure correspondant sous la forme "HH:mm:ss"
Motif heure abrégé (4)	
Motif heure long (5)	
Motif date court (6)	Format d'affichage de date correspondant sous la forme "dddd d MMMM yyyy"
Motif date abrégé (7)	
Motif date long (8)	
Séparateur date (13)	Séparateur utilisé dans les formats de dates (ex : "/")
Séparateur heure (14)	Séparateur utilisé dans les formats d'heures (ex : ":")
Position jour date courte (15)	Position du jour, du mois et de l'année dans le format de date court : "1" = à gauche "2" = au milieu "3" = à droite
Position mois date courte (16)	
Position année date courte (17)	
Libellé AM heure système (18)	Libellé additionnel des heures d'avant midi dans les formats sur 12 heures (ex : "Matin")
Libellé PM heure système (19)	Libellé additionnel des heures après midi dans les formats sur 12 heures (ex : "Après-Midi")

Exemple

Sur un chèque rempli mécaniquement, les sommes inscrites sont généralement précédées de "*" afin d'empêcher les fraudes. Si le format d'affichage système standard pour la monnaie est "$ 5,422.33", le format pour les chèques devrait, lui, être du type "$***5432.33" : pas de virgule entre les milliers et pas d'espace entre le symbole $ et le premier chiffre. Le format à utiliser avec la fonction Chaine doit être "$*******.**". Pour le construire par programmation il est nécessaire de connaître le symbole monétaire et le séparateur décimal :

```
LIRE FORMATAGE SYSTEME(Symbole monétaire;$vSymbMon)
LIRE FORMATAGE SYSTEME(Séparateur décimal;$vSepDec)
$MonFormat:="###"+$vSymbMon+"*******"+$vSepDec+"**"
$Résultat:=Chaine(montant;$MonFormat)
```

Référence

CHOIX FORMATAGE.

LISTE DES POLICES (polices)

Paramètre	Type		Description
polices	Tableau	←	Tableau des noms des polices disponibles

Description

La commande LISTE DES POLICES remplit le tableau polices (de type Alpha ou Texte) avec les noms des polices disponibles dans votre système.

Exemple

Dans un formulaire, vous voulez obtenir une liste déroulante qui affiche les polices disponibles dans le système. Ecrivez la méthode suivante pour votre objet liste déroulante :

```
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU ALPHA (63;taPolices;0)
        LISTE DES POLICES(taPolices)
            ` ...
Fin de cas
```

Référence

Nom de police, Numero de police.

Nom de la machine → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom de la machine sur le réseau

Description

La commande Nom de la machine retourne le nom de la machine tel qu'il a été défini dans les paramètres réseau du système d'exploitation.

Exemple

Même si vous n'utilisez pas la version client/serveur de 4D, votre application peut comprendre des services réseaux qui nécessitent que votre machine soit correctement configurée. Dans la Méthode base Sur ouverture de votre application, vous pouvez écrire :

```
Si ( (Nom de la machine="") | (Nom du possesseur=""))
   ... ` Afficher une boîte de dialogue demandant à l'utilisateur de configurer ses
   ` paramètres réseau
Fin de si
```

Référence

Nom du possesseur.

Nom de police (numéro) → Alpha

Paramètre	Type		Description
numéro	Entier long	→	Numéro de la police de laquelle récupérer le nom
Résultat	Alpha	←	Nom de la police

Description

La fonction Nom de police retourne le nom de la police dont le numéro est passé dans le paramètre numéro. Si aucune police de ce numéro n'est disponible, Nom de police retourne une chaîne vide.

Exemples

(1) Pour afficher un objet dans votre formulaire avec la police du système par défaut, écrivez la ligne suivante :

```
` 0 est le numéro de la police du système
CHANGER JEU DE CARACTERES(monObjet;Nom de police(0))
```

(2) Pour afficher un objet dans votre formulaire avec la police de l'application par défaut, écrivez la ligne suivante :

```
` 1 est le numéro de la police de l'application
CHANGER JEU DE CARACTERES(monObjet;Nom de police(1))
```

Référence

LISTE DES POLICES, Numero de police.

Nom du possesseur → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom du possesseur de la machine sur le réseau

Description

La fonction Nom du possesseur retourne le nom du possesseur de la machine, tel qu'il a été défini dans le compte d'utilisateur courant sur la machine.

Exemple

Reportez-vous à l'exemple de la commande Nom de la machine.

Référence

Nom de la machine.

Nombre ecrans

Nombre ecrans → Entier long

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Entier long ←	Nombre d'écrans

Description

Nombre ecrans renvoie le nombre de moniteurs qui sont connectés à votre machine.

Référence

COORDONNEES ECRAN, Ecran principal, Hauteur ecran, Largeur ecran, PROFONDEUR ECRAN.

Numero de police (nomPolice) → Entier long

Paramètre	Type		Description
nomPolice	Alpha	→	Nom de la police de laquelle récupérer le numéro
Résultat	Entier long	←	Numéro de police

Description

La fonction Numero de police retourne le numéro de la police dont le nom est passé dans le paramètre nomPolice. Si aucune police de ce nom n'existe, Numero de police retourne 0.

Exemple

Si certains formulaires dans votre base utilisent une police qui s'appelle “Spéciale”, vous pouvez écrire le code suivant :

```
Si (Numero de police("Spéciale")=0)
   ALERTE("Vous devriez installer la police Spéciale pour que ce formulaire soit
                                                   correctement affiché.")
Fin de si
```

Référence

LISTE DES POLICES, Nom de police.

PROFONDEUR ECRAN (profondeur; couleurs{; écran})

Paramètre	Type		Description
profondeur	Numérique	←	Profondeur de l'écran (nombre de couleurs = 2 ^ profondeur)
couleurs	Numérique	←	1 = écran couleur 0 = écran noir et blanc ou niveaux de gris
écran	Numérique	→	Numéro de l'écran ou écran principal si omis

Description

La commande PROFONDEUR ECRAN retourne dans les paramètres profondeur et couleurs les caractéristiques du moniteur utilisé.

Après l'appel :

• La profondeur de l'écran est retournée dans profondeur. La profondeur élevée en tant que puissance de 2 vous permet de connaître le nombre de couleurs que votre moniteur affiche. Si par exemple votre moniteur est paramétré en 256 couleurs (2^8), la profondeur de votre écran est de 8.

4D fournit les constantes prédéfinies suivantes :

Constante	Type	Valeur
Noir et blanc	Entier long	0
Quatre couleurs	Entier long	2
Seize couleurs	Entier long	4
Deux cent cinquante six coul	Entier long	8
Milliers de couleurs	Entier long	16
Million de couleurs 24 bits	Entier long	24
Million de couleurs 32 bits	Entier long	32

Si le moniteur est configuré pour afficher des couleurs, le paramètre couleur vaut 1. Si le moniteur est configuré pour afficher des niveaux de gris, couleur vaut 0 (zéro). Notez que cette valeur n'a de signification que sous Mac OS.

Les constantes prédéfinies suivantes sont fournies par 4D :

Constante	Type	Valeur
Est en niveaux de gris	Entier long	0
Est en couleurs Entier long 1		

• Le paramètre optionnel écran vous permet de spécifier le numéro du moniteur sur lequel vous souhaitez obtenir des informations. Si vous omettez le paramètre écran, la commande retourne la profondeur de l'écran principal.

Exemple

Votre application affiche de nombreux graphiques en couleurs. Vous pouvez écrire, quelque part dans votre base :

```
PROFONDEUR ECRAN ($vlProf;$vlCouleur)
Si ($vlProf<16)
   ALERTE("Les formulaires seraient plus beaux si l'écran"+" était configuré en millions
                                                                de couleurs.")
Fin de si
```

Référence

FIXER PROFONDEUR ECRAN, Nombre ecrans.

PROPRIETES PLATE FORME (plateForme{; système{; processeur{; langue}}})

Paramètre	Type		Description
plateForme	Numérique	←	2 = Mac OS, 3 = Windows
système	Numérique	←	Dépend de la version que vous utilisez
processeur	Numérique	←	Famille de processeur
langue	Numérique	←	Dépend du système que vous utilisez

Description

La commande PROPRIETES PLATE FORME retourne des informations sur le type de système d'exploitation que vous utilisez, la version et la langue du système d'exploitation ainsi que le processeur installé.
PROPRIETES PLATE FORME retourne ces informations dans les paramètres plateForme, système, processeur et langue.

• plateForme indique le système d'exploitation utilisé. Ce paramètre retourne une des constantes prédéfinies suivantes :

Constante	Type	Valeur
Mac OS	Entier long	2
Windows	Entier long	3

• Les informations retournées dans le paramètre système dépendent de la version de 4e Dimension que vous utilisez.

Version Macintosh

Avec une version Mac OS de 4D, le paramètre système retourne une valeur sur 32 bits (Entier long), dans laquelle le "mot machine haut" est inutilisé et le "mot machine bas" est structuré ainsi :

- L'octet supérieur contient le numéro de version principal,
- L'octet inférieur est composé de deux "nibbles" de 4 bits chacun. Le nibble supérieur est le numéro de mise à jour principal et le nibble inférieur le numéro de mise à jour secondaire. Par exemple : le système 9.0.4 est codé $0904, vous recevrez donc la valeur décimale 2308.

Note : Dans 4D, vous pouvez extraire ces valeurs à l'aide des Opérateurs numériques % (modulo) et \ (division entière) ou des Opérateurs sur les bits.

Utilisez la formule suivante pour connaître le numéro de version principal de Mac OS :

```
PROPRIETES PLATE FORME($vlPlatform;$vlSystem)
$vlResult:=$vlSystem\256
    `Si $vlResult = 8  → vous êtes sous Mac OS 8.x
    `Si $vlResult = 9  → vous êtes sous Mac OS 9.x
    `Si $vlResult = 16  → vous êtes sous Mac OS 10.x
```

Version Windows

Avec une version Windows de 4D, le paramètre système retourne une valeur sur 32 bits (Entier long), structurée ainsi :

Si le bit supérieur vaut 0, vous utilisez Windows NT, Windows 2000, Windows XP ou Windows Vista. S'il vaut 1 (*obsolète*), vous utilisez Windows 95 ou Windows 98.

Note : Le bit supérieur détermine le signe de la valeur Entier long. De ce fait, dans 4D, vous avez simplement besoin de tester la valeur ; si elle est positive, vous êtes sous Windows NT, Windows 2000, Windows XP ou Windows Vista. Vous pouvez également utiliser les Opérateurs sur les bits.

L'octet inférieur fournit le numéro de version principal de Windows. S'il vaut 4, vous utilisez Windows 95, 98 ou Windows NT 4. S'il vaut 5, vous utilisez Windows 2000 ou Windows XP (dans les deux cas, le signe de la valeur indique si vous utilisez Windows NT/2000 ou non). S'il vaut 6, vous utilisez Windows Vista.

L'octet inférieur suivant fournit le numéro de version secondaire de Windows. Sous Windows 95, sa valeur est 0.

Note : Dans 4D, vous pouvez extraire ces valeurs à l'aide des Opérateurs numériques % (modulo) et \ (division entière) ou des Opérateurs sur les bits.

• Le paramètre processeur indique la "famille" du microprocesseur de la machine. Deux valeurs peuvent être renvoyées, disponibles sous forme de constantes :

Constante	Type	Valeur
Compatible Intel	Entier long	586
Power PC	Entier long	406

La combinaison des paramètres plateForme et processeur permet par exemple de savoir sans ambiguïté si la machine utilisée est de type “MacIntel” (plateForme=Mac OS et processeur=Compatible Intel).

• Le paramètre langue permet de connaître la langue courante du système sur lequel est exécutée la base. Voici la liste des codes pouvant être retournés dans ce paramètre, ainsi que leur signification :

Code	Langue
1	Arabe
2	Bulgare
3	Catalan
4	Chinois
5	Tcheque
6	Danois
7	Allemand
8	Grec
9	Anglais
10	Espagnol
11	Finlandais
12	Français
13	Hébreu
14	Hongrois
15	Islandais
16	Italien
17	Japonais
18	Coréen
19	Néerlandais
20	Norvégien
21	Polonais
22	Portugais
24	Roumain
25	Russe
26	Croate
26	Serbe
27	Slovaque
28	Albanais
29	Suédois
30	Thailandais
31	Turc
33	Indonésien
34	Ukrainien
35	Bélarusse
36	Slovène
37	Estonien

38	Letton
39	Lithuanien
41	Farsi
42	Vietnamien
45	Basque
54	Afrikaans
56	Féroïen

Note : Si la commande n'a pas pu identifier la langue du système, la valeur 9 (Anglais) est retournée.

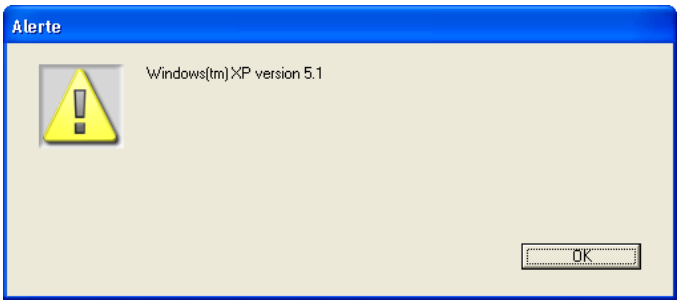
Exemple

La méthode projet suivante affiche une boîte de dialogue d'alerte décrivant le système d'exploitation utilisé :

```
      ` Méthode projet VERSION SYSTEME

PROPRIETES PLATE FORME($vlPlatform;$vlSystem;$vlMachine)
Si (($vlPlatform<2) | ($vlPlatform>3))
   $vsPlatformOS:=""
Sinon
   Si ($vlPlatform=Windows)
      $vsPlatformOS:=""
      Si ($vlSystem<0)
         $winMajVers:=((2^31)+$vlSystem)%256
         $winMinVers:=(((2^31)+$vlSystem)\256)%256
         Si ($winMinVers=0)
            $vsPlatformOS:="Windows™ 95"
         Sinon
            $vsPlatformOS:="Windows™ 98"
         Fin de si
      Sinon
         $winMajVers:=$vlSystem%256
         $winMinVers:=($vlSystem\256)%256
         Au cas ou
            : ($winMajVers=4)
               $vsPlatformOS:="Windows™ NT"
```

```
                : ($winMajVers=5)
                    Si ($winMinVers=0)
                        $vsPlatformOS:="Windows™ 2000"
                    Sinon
                        $vsPlatformOS:="Windows™ XP"
                    Fin de si
                : ($winMajVers=6)
                    $vsPlatformOS:="Windows™ Vista"

            Fin de cas
        Fin de si
        $vsPlatformOS:=$vsPlatformOS+" version "+Chaine($winMajVers)+"."+
                                                        Chaine($winMinVers)
    Sinon
        $vsPlatformOS:="Mac OS™ version "
        Si (($vlSystem\256) = 16)
            $vsPlatformOS:=$vsPlatformOS+"10"
        Sinon
            $vsPlatformOS:=$vsPlatformOS+Chaine($vlSystem\256)
        Fin de si
        $vsPlatformOS:=$vsPlatformOS+"."+Chaine(($vlSystem\16)%16)+
                                ("."+Chaine($vlSystem%16))*Num(($vlSystem%16) # 0))
    Fin de si
Fin de si
ALERTE($vsPlatformOS)
```

Sous Windows, vous obtenez une boîte de dialogue semblable à celle-ci :



Sous Mac OS, vous obtenez une boîte de dialogue semblable à celle-ci :

Référence

Opérateurs sur les bits.

Selectionner couleur RVB ({coulDefaut}{; }{message}) → Entier long

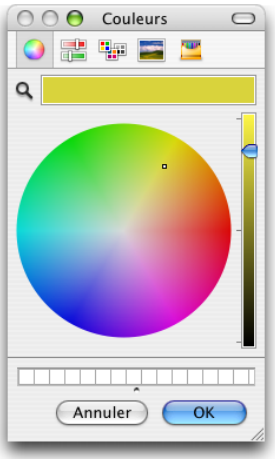
Paramètre	Type		Description
coulDefaut	Entier long	→	Couleur RVB présélectionnée
message	Alpha 255	→	Titre de la fenêtre de sélection
Résultat	Entier long	←	Couleur RVB

Description

La commande Selectionner couleur RVB affiche la fenêtre système de sélection de couleur et retourne la valeur RVB de la couleur sélectionnée par l'utilisateur.

La fenêtre système de sélection de couleur a l'apparence suivante :

Macintosh



Windows

Le paramètre facultatif coulDéfaut vous permet de pré-sélectionner une couleur dans la fenêtre. Ce paramètre vous permet par exemple de restituer par défaut la dernière couleur définie par l'utilisateur. Passez dans ce paramètre une valeur de couleur au format RVB (pour plus d'informations, reportez-vous à la description de la commande FIXER COULEURS RVB). Vous pouvez utiliser l'une des constantes du thème "FIXER COULEURS RVB". Si le paramètre coulDéfaut est omis ou si vous passez 0, la couleur noir est sélectionnée à l'ouverture de la boîte de dialogue.

Le paramètre facultatif message vous permet de personnaliser le titre de la fenêtre système. Par défaut, si ce paramètre est omis, le libellé “Couleurs” est affiché.

Si l’utilisateur valide la boîte de dialogue, la commande retourne la valeur de couleur sélectionnée au format RVB et la variable système *OK* prend la valeur 1. Si l’utilisateur annule la boîte de dialogue, la commande retourne -1 et la variable système *OK* prend la valeur 0.

Note : Cette commande ne doit pas être exécutée sur le poste serveur ni dans le cadre d’un process Web.

Référence

FIXER COULEURS RVB.

Variables et ensembles système

Si l’utilisateur valide la boîte de dialogue, la variable système *OK* prend la valeur 1, sinon elle prend la valeur 0.

18

Etats rapides

QR APPELER SUR COMMANDE

QR APPELER SUR COMMANDE (zone; nomMéthode)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
nomMéthode	Alpha	→	Nom de la méthode à appeler

Description

La commande QR APPELER SUR COMMANDE exécute la méthode projet 4D dont le nom est passé dans le paramètre nomMéthode lorsqu'une commande de l'éditeur d'états rapides est appelée via la sélection d'un menu ou le clic sur un bouton.

Note : Cette commande ne fonctionne pas avec les fenêtres externes en mode Développement.

Si le paramètre zone vaut 0 (zéro), la méthode nomMéthode sera appelée pour toutes les zones de l'éditeur d'états rapides jusqu'à ce que la base soit refermée ou que l'instruction suivante soit exécutée : QR APPELER SUR COMMANDE(0;"").

La méthode nomMéthode reçoit deux paramètres :
• $1 contient la référence de la zone (Entier long).
• $2 contient le numéro de la commande sélectionnée (Entier long).

Note : Si vous souhaitez compiler votre base à l'aide du Compilateur, vous devez déclarer explicitement les paramètres $1 et $2 en entiers longs, même si vous ne les utilisez pas.

Si vous souhaitez que la commande initiale choisie par l'utilisateur soit exécutée, utilisez l'instruction suivante dans la méthode nomMéthode : QR EXECUTER COMMANDE($1;$2).

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR EXECUTER COMMANDE, QR Lire statut commande.

QR BLOB VERS ETAT (zone; blob)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
blob	BLOB	→	BLOB contenant l'état

Description

La commande QR BLOB VERS ETAT place l'état contenu dans le paramètre blob dans la zone d'état rapide désignée par le paramètre zone.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre blob est incorrect, l'erreur -9852 est générée.

Exemples

(1) Le code suivant affiche dans la zone *MaZone* l'état rapide "etat.4qr", stocké à côté du fichier de structure de la base. A noter que le fichier d'état peut avoir été créé avec une version de 4D antérieure à la 2003 :

```
C_BLOB($doc)
C_ENTIER LONG(MaZone)
DOCUMENT VERS BLOB("etat.4qr";$doc)
QR BLOB VERS ETAT (MaZone;$doc)
```

(2) L'instruction suivante affiche l'état stocké dans le champ *ChampBlob* dans la zone *MaZone* :

```
QR BLOB VERS ETAT (MaZone;[Table 1]ChampBlob)
```

Référence

QR ETAT VERS BLOB.

QR Chercher colonne (zone; expression) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
expression	Chaîne | Pointeur	→	Objet de colonne
Résultat	Entier long	←	Numéro de colonne

Description

La commande QR Chercher colonne retourne le numéro de la première colonne de la zone dont le contenu correspond à l'expression passée en paramètre.

expression peut contenir soit une chaîne soit un pointeur.

QR Chercher colonne retourne -1 si la recherche n'aboutit pas.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Exemple

Le code suivant permet de récupérer le numéro de la colonne contenant le champ [G.ER Tests]Quarter puis de supprimer la colonne :

```
$NumColonne:=QR Chercher colonne (MaZone;->[G.ER Tests]Quarter)
```

ou :

```
$NumColonne:=QR Chercher colonne (MaZone;"[G.ER Tests]Quarter")

Si ($NumColonne#-1)
    QR SUPPRIMER COLONNE (MaZone; $NumColonne)
Fin de si
```

QR Creer zone hors ecran

QR Creer zone hors ecran → Entier long

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Entier long ←	Référence de la zone créée

Description

La commande QR Creer zone hors ecran crée une zone d'Etat rapide hors écran et retourne son numéro de référence.

Référence

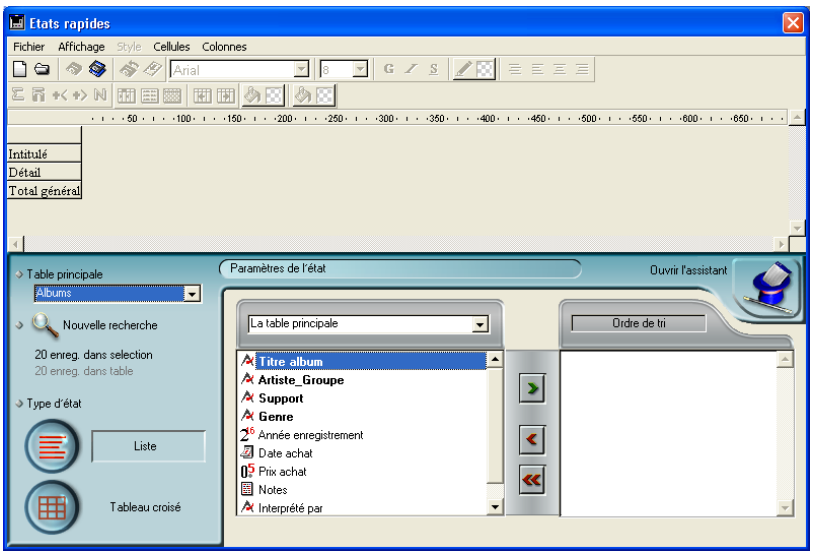
QR SUPPRIMER ZONE HORS ECRAN.

QR ETAT ({table; }document{; hiérarchique{; assistant{; recherche{; *}}}})

Paramètre	Type		Description
table	Table	→	Table à utiliser ou Table par défaut si ce paramètre est omis
document	Alpha	→	Document d'état rapide à charger
hiérarchique	Booléen	→	Vrai = Afficher les tables N liées Faux ou omis = Ne pas les afficher
assistant	Booléen	→	Vrai = Afficher le bouton de l'assistant Faux ou omis = Ne pas l'afficher
recherche	Booléen	→	Vrai = Afficher les outils de recherche et la table principale; Faux ou omis = Ne pas les afficher
*	*	→	Suppression des boîtes de dialogue d'impression

Description

La commande QR ETAT imprime un état pour table, à l'aide de l'Editeur d'états rapides présenté ci-dessous.



Cet éditeur permet à l'utilisateur de construire en totalité son propre état. Pour plus d'informations sur la création d'états à l'aide de l'Editeur d'états rapides, reportez-vous au manuel *Mode Développement* de 4D.

Notes :
• L'éditeur n'apparaît pas si la table a été déclarée “Invisible”.
• Lorsque l'éditeur est appelé via la commande QR ETAT, l'option **Tous les liens en automatique**, permettant de modifier le statut automatique/manuel des liens, est masquée. Ce principe permet au développeur de gérer lui-même ce statut à l'aide des commandes FIXER LIENS AUTOMATIQUES et FIXER LIEN CHAMP.

• document
Le paramètre document désigne un modèle d'état créé dans l'éditeur d'états rapides et sauvegardé sur disque. Le document stocke les paramètres de l'état, pas les enregistrements. Si une chaîne vide ("") est passée dans document, QR ETAT affiche une boîte de dialogue d'ouverture de fichiers, dans laquelle l'utilisateur peut choisir un modèle d'état à imprimer.
Si le paramètre document spécifie un document qui n'existe pas (si vous passez, par exemple, Caractere (1) dans document), l'éditeur d'états rapides s'affiche.

• hiérarchique
Le paramètre hiérarchique indique si les tables liées N doivent être ou non affichées dans la liste de sélection de champs. Par défaut, sa valeur est 0 (les tables N ne sont pas affichées).

• assistant
Ce paramètre permet d’indiquer si le bouton **Ouvrir l'assistant** doit apparaître ou non dans la fenêtre de l’éditeur d’états rapides. Passez Vrai pour afficher le bouton et Faux pour le masquer. Par défaut (si ce paramètre est omis), le bouton n'est pas affiché.

• recherche
Ce paramètre permet d’indiquer si le bouton **Nouvelle recherche** doit apparaître ou non dans la fenêtre de l’éditeur d’états rapides. Passez Vrai pour afficher le bouton et Faux pour le masquer.
Par défaut (si ce paramètre est omis), le bouton n'est pas affiché.

Une fois qu'un fichier d'état est sélectionné, les boîtes de dialogue d'impression s'affichent, sauf si le paramètre * a été spécifié — dans ce cas, elles ne s'affichent pas. L'état est alors imprimé.

Lorsque l'Editeur d'états rapides n'est pas affiché, la variable système OK prend la valeur 1 si un état est imprimé ; sinon elle prend la valeur 0 (zéro) — par exemple si l'utilisateur a cliqué sur **Annuler** dans les boîtes de dialogue d'impression.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :
• Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande avec le paramètre * ou >.
• La syntaxe faisant apparaître l'éditeur d'états rapide ne fonctionne pas avec 4D Server, dans ce cas la variable système OK prend la valeur 0.
• En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemples

(1) L'exemple suivant permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis imprime automatiquement l'état "Liste détaillée" :

```
CHERCHER ([Personnes])
Si (OK=1)
    QR ETAT ([Personnes]; "Liste détaillée";Faux;Faux;Faux;*)
Fin de si
```

(2) L'exemple suivant permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis de sélectionner le document d'état qui sera ensuite utilisé pour l'impression :

```
CHERCHER ([Personnes])
Si (OK=1)
    QR ETAT ([Personnes];"";Faux;Faux;Faux)
Fin de si
```

(3) L'exemple suivant permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis affiche l'Editeur d'états rapides afin que l'utilisateur puisse construire, charger, sauvegarder ou imprimer tout état, avec ou sans l'assistant :

```
CHERCHER ([Personnes])
Si (OK=1)
    QR ETAT ([Personnes]; Caractere(1);Faux;Vrai)
Fin de si
```

(4) Reportez-vous à l'exemple de la commande FIXER LIEN CHAMP.

Référence

FIXER METHODES AUTORISEES, IMPRIMER ETIQUETTES, IMPRIMER SELECTION.

QR ETAT VERS BLOB

QR ETAT VERS BLOB (zone; blob)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
blob	BLOB	→	BLOB devant recevoir l'état rapide

Description

La commande QR ETAT VERS BLOB place dans le BLOB blob (variable ou champ) l'état dont la référence a été passée dans le paramètre zone.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Exemple

L'instruction suivante affecte l'état rapide stocké dans la zone *MaZone* à un champ BLOB :

```
QR ETAT VERS BLOB (MaZone;[Table 1]ChampBlob)
```

Référence

QR BLOB VERS ETAT.

QR EXECUTER (zone)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone à exécuter

Description

La commande QR EXECUTER provoque l'exécution de l'état rapide désigné par le paramètre zone. L'état est généré avec ses paramétrages courants, notamment son type de sortie. Vous pouvez utiliser la commande QR FIXER DESTINATION pour modifier le type de sortie.

L'état est exécuté sur la table à laquelle appartient la zone. Lorsque zone désigne une zone hors écran, il est nécessaire de spécifier la table à utiliser à l'aide de la commande QR FIXER TABLE ETAT.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

QR EXECUTER COMMANDE (zone; numCommande)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numCommande	Entier long	→	Commande de menu à exécuter

Description

La commande QR EXECUTER COMMANDE exécute la commande de menu ou le bouton de la barre d'outils dont la référence est passée dans le paramètre numCommande. En général, cette commande est utilisée pour exécuter une commande de menu sélectionnée par l'utilisateur et interceptée dans votre code via la commande QR APPELER SUR COMMANDE.

Passez dans paramètre numCommande une des constantes du thème QR Commandes.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numCommande est incorrect, l'erreur -9852 est générée.

Référence

QR APPELER SUR COMMANDE, QR Lire statut commande.

QR FIXER DESTINATION

QR FIXER DESTINATION (zone; type{; spécificités})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
type	Entier long	→	Type d'état
spécificités	Chaîne | Variable	→	Spécificités du type de destination

Description

La commande QR FIXER DESTINATION permet de définir le type de destination de sortie de l'état rapide contenu dans la zone.

Passez dans le paramètre type une des constantes du thème QR Destination de sortie. Le contenu du paramètre spécificités dépend de la valeur de type. Le tableau suivant liste les valeurs qui peuvent être passées dans les paramètres type et spécificités.

Destination	type (valeur)	spécificités
Imprimante	qr imprimante (1)	N.A.
Fichier texte	qr fichier texte (2)	Chemin d'accès du fichier
4D View	qr zone 4D View (3)	N.A.
4D Chart	qr zone 4D Chart (4)	N.A.
Fichier HTML	qr fichier HTML (5)	Chemin d'accès du fichier

qr fichier texte (2) : Si vous passez une chaîne vide dans le paramètre spécificités, une boîte de dialogue standard d'enregistrement de fichiers apparaît. Si vous passez un chemin d'accès valide, l'état rapide sera enregistré à l'emplacement indiqué.
Par défaut, le délimiteur de champ est le caractère Tabulation (code 9) et le délimiteur d'enregistrement est le caractère Retour chariot (code 13). Vous pouvez modifier ces caractères par défaut en changeant la valeur des variables système *FldDelimit* et *RecDelimit*. Sous Windows, si *FldDelimit* vaut 13, un caractère 10 (Saut de ligne) sera ajouté après le Retour chariot. Tenez compte du fait que ces variables sont utilisées par d'autres commandes, par exemple IMPORTER TEXTE. Toute modification de ces variables est répercutée sur l'ensemble de l'application.

qr zone 4D View (3) : Si l'utilisateur courant dispose du plug-in 4D View, une fenêtre externe 4D View est créée et affiche les résultats des paramètres courants de la zone d'état rapide.

qr zone 4D Chart (4) : une fenêtre externe 4D Chart est créée et affiche les résultats des paramètres courants de la zone d'état rapide. Pour plus d'informations sur le mode de conversion des données, veuillez vous référer au manuel *Mode Développement*.

qr fichier HTML (5) : Un fichier HTML est généré d'après les paramètres courants de la zone d'état rapide. Le fichier HTML est basé sur le modèle défini par la commande QR FIXER MODELE HTML. Pour plus d'informations sur le mode de conversion des données, veuillez vous référer au manuel *Mode Développement*.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si la valeur de type de destination est incorrecte, l'erreur -9852 est générée.

Exemple

L'exemple suivant définit le fichier texte "MonDoc.txt" comme type de destination de l'état puis l'exécute :

```
QR FIXER DESTINATION(MaZone; qr fichier texte; "MonDoc.txt")
QR EXECUTER(MaZone)
```

Référence

QR LIRE DESTINATION.

QR FIXER DONNEES TOTAUX

QR FIXER DONNEES TOTAUX (zone; numColonne; numRupture; opérateur | valeur)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne
numRupture	Entier long	→	Numéro de rupture
opérateur | valeur	Entier long | Chaîne	→	Opérateur pour la cellule ou Contenu de la cellule

Description

Note : Cette commande ne crée pas de sous-total.

Etat en liste
La commande QR FIXER DONNEES TOTAUX permet de définir le contenu d'une ligne de rupture spécifique (sous-total ou total général).

Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de colonne de la cellule que vous souhaitez définir.

Passez dans numRupture le numéro de la ligne de rupture à modifier (sous-total ou total général). Pour une ligne de sous-total, numRupture correspond au numéro d'ordre de la rupture. Pour le total général, numRupture vaut -3 (vous pouvez également utiliser la constante qr total général du thème QR Lignes pour Propriétés).

Le paramètre opérateur contient la valeur cumulée de tous les opérateurs à placer dans la cellule. Utilisez les constantes du thème QR Opérateurs pour définir ce paramètre :

Constante	Valeur
qr somme	1
qr moyenne	2
qr min	4
qr max	8
qr nombre	16
qr écart type	32

Si vous ne souhaitez utiliser aucun opérateur, passez 0 dans le paramètre opérateur.

Si vous choisissez d'insérer du texte dans la cellule, passez-le dans le paramètre valeur.

Note : Les paramètres opérateur et valeur sont mutuellement exclusifs, vous pouvez passer soit une combinaison de valeurs numériques, soit du texte.

Si vous souhaitez saisir à la fois du texte et des opérateurs, vous pouvez utiliser les codes suivants dans le paramètre valeur :
- # pour la valeur provoquant la rupture ou le sous-total
- ##S sera remplacé par la somme.
- ##A sera remplacé par la moyenne.
- ##C sera remplacé par le nombre
- ##X sera remplacé par le maximum.
- ##N sera remplacé par le minimum.
- ##D sera remplacé par l'écart type.
- ##xx, où xx est un numéro de colonne. Ce code sera remplacé par la valeur de la colonne désignée, dans son propre formatage. Si la colonne n'existe pas, le code apparaît dans l'état.

Etat tableau croisé
La commande QR FIXER DONNEES TOTAUX vous permet de définir le contenu d'une cellule spécifique.

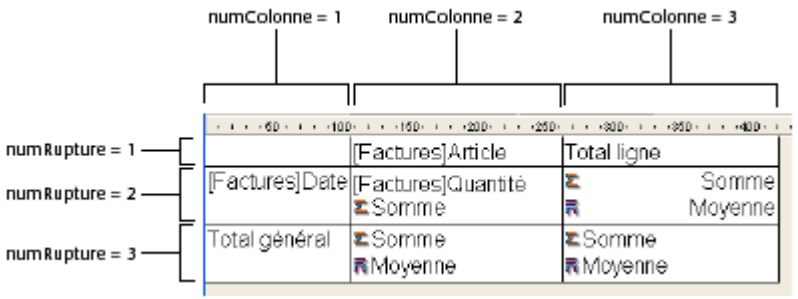
Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de colonne de la cellule que vous souhaitez définir.

Passez dans numRupture le numéro de ligne de la cellule que vous souhaitez définir.

Le paramètre opérateur contient la valeur cumulée de tous les opérateurs à placer dans la cellule. Utilisez les constantes du thème QR Opérateurs pour définir ce paramètre (cf. paragraphe précédent).
Le paramètre alternatif valeur permet de définir le texte à insérer dans la cellule.

L'illustration suivante précise la manière dont les paramètres numColonne et numRupture sont combinés dans un tableau croisé :



Types de données acceptés
Vous pouvez passer deux types de données : des libellés et des opérateurs.

• Libellés
Un libellé est une chaîne de caractères passée via le paramètre valeur. Cette valeur ne peut être utilisée qu'avec les cellules suivantes : numColonne=3,numRupture=1 et numColonne=1,numRupture=3.

• Opérateurs
Un opérateur ou un cumul d'opérateurs (cf. paragraphe précédent) peut être passé via le paramètre opérateur aux cellules suivantes :
numColonne=2,numRupture=2
numColonne=3,numRupture=2
numColonne=2,numRupture=3
Notez que ces deux dernières valeurs affectent également la cellule (colonne 3,ligne 3). En effet, si par exemple un calcul est effectué dans la cellule (colonne 2,ligne 3), le contenu de la cellule (colonne 3/ligne 3) sera modifié en conséquence.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.
Si le paramètre numRupture est incorrect, l'erreur -9853 est générée.

Référence

QR LIRE DONNEES TOTAUX.

QR FIXER ENCADREMENTS

QR FIXER ENCADREMENTS (zone; colonne; ligne; encadrements; épaisseur{; couleur})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
colonne	Entier long	→	Numéro de colonne
ligne	Entier long	→	Numéro de ligne
encadrements	Entier long	→	Valeur d'encadrements composée
épaisseur	Entier long	→	Epaisseur de ligne
couleur	Entier long	→	Couleur de ligne

Description

La commande QR FIXER ENCADREMENTS permet de définir le style d'encadrement d'une cellule spécifique.

Passez dans zone la référence de la zone d'état rapide.

Passez dans colonne le numéro de colonne de la cellule à encadrer.

Le paramètre ligne contient le numéro de ligne de la cellule à encadrer :
• passez -1 pour désigner l'intitulé de l'état,
• passez -2 pour désigner la zone Détail de l'état,
• passez -3 pour désigner la zone Total général,
• passez une valeur entière positive pour désigner la ligne de sous-total correspondante.
Vous pouvez également utiliser les constantes correspondantes dans le thème QR Lignes pour Propriétés.

Le paramètre encadrement contient une valeur composée permettant d'indiquer la ou les bordures(s) de cellule à modifier :
• 1 désigne la bordure gauche,
• 2 désigne la bordure supérieure,
• 4 désigne la bordure droite,
• 8 désigne la bordure inférieure,
• 16 désigne la bordure intérieure verticale,
• 32 désigne la bordure intérieure horizontale.
Vous pouvez également utiliser les constantes correspondantes dans le thème QR Encadrements.

Le paramètre encadrement peut contenir un cumul de plusieurs valeurs afin de désigner simultanément plusieurs bordures. Par exemple, si vous passez 5 dans encadrement, les bordures droite et gauche seront affectées.

Le paramètre épaisseur permet de spécifier l'épaisseur de l'encadrement :
• 0 indique une épaisseur nulle (pas de bordure),
• 1 indique une épaisseur d'1/4 point,
• 2 indique une épaisseur d'1/2 point,
• 3 indique une épaisseur d'1 point,
• 4 indique une épaisseur de 2 points.

Le paramètre couleur permet de désigner la couleur de l'encadrement :
• si vous passez une valeur positive, couleur désigne un numéro de couleur,
• si vous passez 0, la couleur est noire,
• si vous passez -1, la couleur esrt inchangée.
Note : Par défaut, la couleur est noire.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre colonne est incorrect, l'erreur -9852 est générée.
Si le paramètre ligne est incorrect, l'erreur -9853 est générée.
Si le paramètre encadrement est incorrect, l'erreur -9854 est générée.
Si le paramètre épaisseur est incorrect, l'erreur -9855 est générée.

Référence

QR LIRE ENCADREMENTS.

QR FIXER ENTETE ET PIED DE PAGE (zone; sélecteur; titreGauche; titreCentre; titreDroite; hauteur{; image{; alignementImage}})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
sélecteur	Entier long	→	1 = En-tête, 2 = Pied de page
titreGauche	Alpha	→	Texte affiché sur le côté gauche
titreCentre	Alpha	→	Texte affiché au centre
titreDroite	Alpha	→	Texte affiché sur le côté droit
hauteur	Numérique	→	Hauteur de l'en-tête ou du pied de page
image	Image	→	Image à afficher
alignementImage	Entier long	→	Alignement de l'image

Description

La commande QR FIXER ENTETE ET PIED DE PAGE vous permet de définir le contenu et la taille de l'en-tête et du pied de page de la zone.

Le paramètre sélecteur vous permet de définir la zone à affecter :
• si sélecteur vaut 1, l'en-tête sera affecté ;
• si sélecteur vaut 2, le pied de page sera affecté.

Les paramètres titreGauche, titreCentre et titreDroite définissent le contenu des en-têtes ou pieds de page situés respectivement à gauche, au centre et à droite.

Le paramètre hauteur indique la hauteur de l'en-tête ou du pied de page, exprimée dans l'unité courante de l'état rapide.

Le paramètre image contient l'image à afficher facultativement dans l'en-tête ou le pied de page.

Le paramètre alignementImage définit la propriété d'alignement de l'image passée dans image :
• si alignementImage vaut 0, l'image est alignée sur la gauche.
• si alignementImage vaut 1, l'image est centrée.
• si alignementImage vaut 2, l'image est alignée sur la droite.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre sélecteur est incorrect, l'erreur -9852 est générée.

Exemple

L'instruction suivante place le libellé "Titre du centre" dans l'en-tête de l'état rapide dans *MaZone* et définit sa hauteur à 200 points :

```
QR FIXER ENTETE ET PIED DE PAGE(MaZone; 1;"";"Titre du centre";""; 200)
```

Référence

QR LIRE ENTETE ET PIED DE PAGE.

QR FIXER ESPACEMENT TOTAUX

QR FIXER ESPACEMENT TOTAUX (zone; sousTotal; valeur)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
sousTotal	Entier long	→	Numéro de sous-total
valeur	Entier long	→	0=pas d'espace, 32000=insère une saut de page, >0=espace ajouté en haut du niveau de rupture, <0=augmentation proportionnelle

Description

La commande QR FIXER ESPACEMENT TOTAUX permet de définir l'espacement ajouté au-dessus d'une ligne de sous-total. Elle ne peut être utilisée qu'avec des états en liste.

Le paramètre zone contient la référence de la zone d'état rapide.
Le paramètre sousTotal désigne le niveau de sous-total (ou de rupture) à modifier.

Le paramètre valeur permet de définir la valeur de l'espacement :
• Si valeur vaut 0, aucun espacement n'est ajouté.
• Si valeur vaut 32000, un saut de page est ajouté.
• Si valeur est une valeur positive, elle exprime l'espacement à ajouter en pixels.
• Si valeur est une valeur négative, elle exprime l'espacement à ajouter en pourcentage de la ligne de sous-total. Par exemple, la valeur -100 définit l'ajout d'un espace au-dessus de la ligne de sous-total correspondant à 100% de la hauteur courante de la ligne.

Note : Si l'espacement ajouté au-dessus d'une ligne de sous-total “repousse” la ligne sur la page suivante, aucun espace n'apparaîtra au-dessus de la ligne sur cette page.

Si un numéro de zone invalide est passé, l’erreur -9850 est générée.
Si le paramètre sousTotal est incorrect, l’erreur -9852 est générée.

Référence

QR LIRE ESPACEMENT TOTAUX.

QR FIXER INFO COLONNE

QR FIXER INFO COLONNE (zone; numColonne; titre; objet; cachée; taille; valeursRépétées; format)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne
titre	Alpha	→	Titre de la colonne
objet	Champ | Variable	→	Objet affecté à la colonne
cachée	Entier long	→	0 = visible, 1 = invisible
taille	Entier long	→	Largeur de la colonne
valeursRépétées	Entier long	→	0 = Non répétées, 1 = Répétées
format	Alpha	→	Format d'affichage

Description

Etats en liste
La commande QR FIXER INFO COLONNE vous permet de définir les paramètres d'une colonne existante de l'état présent dans la zone.

Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de la colonne à définir.

Passez dans titre l'intitulé devant apparaître dans l'en-tête de la colonne.

Passez dans objet la référence de l'objet devant être affecté à la colonne (variable, champ ou formule).

Le paramètre cachée indique si la colonne doit être affichée ou masquée :
• si cachée vaut 1, la colonne est masquée ;
• si cachée vaut 0, la colonne est affichée.

Passez dans taille la taille en pixels à assigner à la colonne. Si taille vaut -1, la taille de la colonne est automatique.

valeursRépétées indique le statut de la propriété de répétition des données. Par exemple, si la valeur d'un champ ou d'une variable ne change pas d'un enregistrement à l'autre, il est possible de la répéter ou non dans chaque ligne de la colonne.
• si valeursRépétées vaut 0, les valeurs ne sont pas répétées.
• si valeursRépétées vaut 1, les valeurs sont répétées.

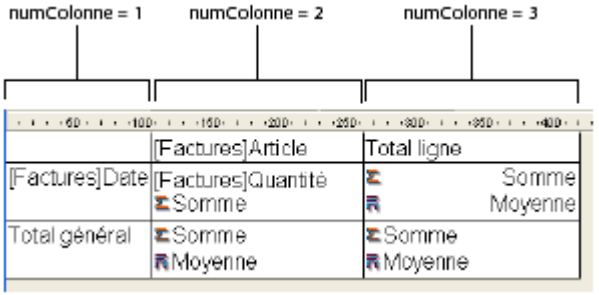
Le paramètre format indique le format d'affichage de la colonne. Vous pouvez utiliser tout format d'affichage standard de 4D compatible avec les données affichées dans la colonne.

Exemple :
La ligne suivante associe à la colonne 1 l'intitulé “Titre” et le champ Champ2, rend la colonne visible avec une largeur de 150 pixels et définit le format d'affichage ###,##.

QR FIXER INFO COLONNE(zone; 1;"Titre"; "[Table 1]Champ2";0;150;0;"###,##")

Etats tableaux croisés
Avec ce type d'état, la commande QR FIXER INFO COLONNE permet de définir globalement les mêmes paramètres que ceux décrits dans le paragraphe ci-dessus, toutefois les zones auxquelles ils s'appliquent sont différentes et varient en fonction du paramètre à fixer.
En outre, les paramètres titre, cachée et valeursRépétées ne sont pas utilisés lorsque vous travaillez avec des états tableaux croisés. La valeur à passer dans le paramètre numColonne dépend de l'opération que vous souhaitez effectuer : définir la taille de la colonne ou définir la source de données et le format d'affichage.

• *Taille de la colonne*
Il s'agit d'un attribut “visuel”, par conséquent les colonnes sont simplement numérotées de gauche à droite, comme illustré ci-dessous :

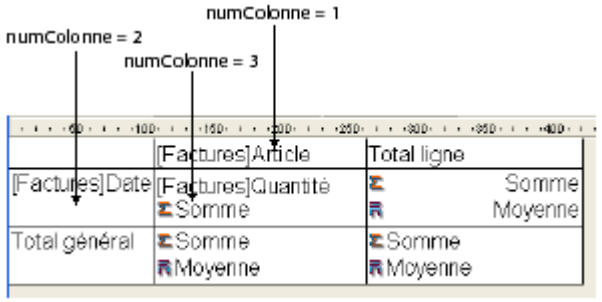


La méthode suivante affecte une taille automatique à toutes les colonnes d'un état en tableau croisé et laisse les autres éléments inchangés :

```
Boucle ($i;1;3)
    QR LIRE INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;$taille;$rep;$format)
    QR FIXER INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;0;$rep;$format)
Fin de boucle
```

A noter que, comme vous voulez modifier uniquement la taille de la colonne, vous devez utiliser la commande QR LIRE INFO COLONNE pour récupérer les propriétés courantes de la colonne puis les passer à QR FIXER INFO COLONNE afin de les conserver inchangées, excepté pour la taille.

• *Source de données (objet) et format d'affichage*
Dans ce cas, la numérotation des colonnes s'effectue de la manière suivante :

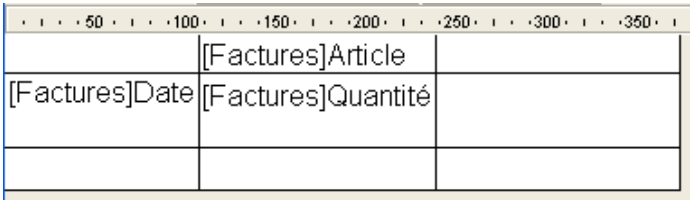


A noter qu'il n'est pas possible d'adresser toutes les cellules avec la commande QR FIXER INFO COLONNE, les cellules non numérotées dans le schéma ci-dessus doivent être gérées à l'aide de la commande QR FIXER DONNEES TOTAUX.

Le code suivant associe des sources de données aux trois cellules nécessaires à la construction d'un état en tableau croisé simple :

```
QR FIXER TABLE ETAT(qr_zone;Table(->[Factures]))
TOUT SELECTIONNER([Factures])
QR FIXER TYPE ETAT(qr_zone;2)
QR FIXER INFO COLONNE(qr_zone;1;"";->[Factures]Article;1;-1;1;"")
QR FIXER INFO COLONNE(qr_zone;2;"";->[Factures]Date;1;-1;1;"")
QR FIXER INFO COLONNE(qr_zone;3;"";->[Factures]Quantité;1;-1;1;"")
```

La zone d'état suivante est générée :



Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.

Référence

QR FIXER INFO LIGNE, QR LIRE INFO COLONNE, QR Lire info ligne.

QR FIXER INFO LIGNE

QR FIXER INFO LIGNE (zone; ligne; cachée)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
ligne	Entier long	→	Ligne
cachée	Entier long	→	0 = Visible, 1 = Cachée

Description

La commande QR FIXER INFO LIGNE permet d'afficher ou de masquer la ligne dont la référence est passée dans le paramètre ligne.

Le paramètre ligne désigne la ligne à modifier :
• passez une valeur entière positive pour désigner la ligne de sous-total correspondante,
• passez -1 pour désigner l'intitulé de l'état,
• passez -2 pour désigner la zone Détail de l'état,
• passez -3 pour désigner la zone Total général,
Vous pouvez également utiliser l'une des trois constantes correspondantes dans le thème QR Lignes pour Propriétés.

cachée permet de spécifier si le contenu de la ligne doit être affiché ou masqué :
• si cachée vaut 1, le contenu de la ligne est masqué ;
• si cachée vaut 0, le contenu de la ligne est affiché.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre ligne est incorrect, l'erreur -9852 est générée.

Exemple

L'instruction suivante masque le contenu de la ligne Détail :

```
QR FIXER INFO LIGNE (maZone;qr détail; 1)
```

Référence

QR FIXER INFO COLONNE, QR LIRE INFO COLONNE, QR Lire info ligne.

QR FIXER MODELE HTML (zone; modèle)

Paramètre	Type		Description
zone	Entier long	-->	Référence de la zone
modèle	Texte	-->	Code du modèle HTML

Description

La commande QR FIXER MODELE HTML permet de définir le modèle HTML à utiliser pour la zone d'état rapide référencée par zone. Ce modèle sera utilisé lors de l'exécution des états au format HTML.

Le modèle est construit à l'aide d'un ensemble de balises de traitement des données. Ce fonctionnement vous permet de générer des documents HTML proches des états originaux ou des documents à l'apparence entièrement personnalisée.

Note : Vous devez appeler au préalable QR FIXER DESTINATION pour définir le format HTML comme destination de sortie.

Balises HTML

<!--#4DQRheader--> ... <!--/#4DQRheader-->
Les intitulés des colonnes seront insérés entre ces balises. Ces balises sont généralement utilisées pour définir la ligne de titre de l'état.

<!--#4DQRrow--> ... <!--/#4DQRrow-->
Les informations insérées entre ces balises seront répétées pour chaque ligne de données (détail et sous-total compris).

<!--#4DQRcol--> ... <!--/#4DQRcol-->
Les informations insérées entre ces balises seront répétées pour chaque colonne de données à l'intérieur des lignes. Le tri de la colonne est identique à celui de l'état. Lorsqu'elles sont utilisées conjointement à <!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->, les balises <!--#4DQRcol--> ... <!--/#4DQRcol--> ne seront effectives qu'avec les colonnes dont le contenu n'est pas inséré à l'aide de <!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->.

Par exemple, dans un état comportant cinq colonnes, vous utilisez les balises <!--#4DQRcol;2--> ... <!--/#4DQRcol;2--> afin d'insérer les données de la deuxième colonne. <!--#4DQRcol--> ... <!--/#4DQRcol--> traiteront, pour chaque ligne, les colonnes 1, 3, 4 et 5. Ces balises ignoreront la colonne dont le contenu est publié à l'aide de <!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->.

<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->
Les informations insérées entre ces balises seront extraites de la colonne de l'état dont le numéro est “n”. Si, par exemple, dans un état HTML à trois colonnes, vous souhaitez afficher les colonnes dans un ordre différent de celui de l'état initial, vous pouvez écrire :
<!--#4DQRrow--> <!--#4DQRcol;3--> ... <!--/#4DQRcol;3--><!--#4DQRcol;2--> ...
<!--/#4DQRcol;2--><!--#4DQRcol;1--> ... <!--/#4DQRcol;1--> <!--/#4DQRrow-->
Dans cet exemple, les colonnes sont générées dans l'ordre inverse de l'état.

<!--#4DQRfont--> ... <!--/#4DQRfont-->
Les informations insérées entre ces balises seront utilisées pour la définition de la police de la colonne ou cellule courante.
<!--#4DQRfont--> sera remplacé par une définition de police HTML et <!--/#4DQRfont--> sera remplacé par la balise de fermeture standard ().

<!--#4DQRface--> ... <!--/#4DQRface-->
Les informations insérées entre ces balises seront utilisées pour la définition du style de la colonne ou cellule courante.
<!--#4DQRface--> sera remplacé par une définition de style HTML <!--#4DQRface--> sera remplacé par la balise de fermeture standard (</face>).

<!--#4DQRbgcolor-->
Cette balise de couleur sera remplacée par la définition de couleur de la cellule courante.

<!--#4DQRdata-->
Cette balise sera remplacée par les données de la cellule courante.

<!--#4DQRlHeader--><!--#4DQRdata--><!--/#4DQRlHeader-->
<!--#4DQRcHeader--><!--#4DQRdata--><!--/#4DQRcHeader-->
<!--#4DQRrHeader--><!--#4DQRdata--><!--/#4DQRrHeader-->
Ces balises seront remplacées respectivement par les données de l'en-tête gauche, central et droit.

<!--#4DQRlFooter--><!--#4DQRdata--><!--/#4DQRlFooter-->
<!--#4DQRcFooter--><!--#4DQRdata--><!--/#4DQRcFooter-->
<!--#4DQRrFooter--><!--#4DQRdata--><!--/#4DQRrFooter-->
Ces balises seront remplacées respectivement par les données du pied de page gauche, central et droit.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR Lire modele HTML.

QR FIXER PROPRIETE DOCUMENT

QR FIXER PROPRIETE DOCUMENT (zone; propriété; valeur)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
propriété	Entier long	→	1 = Dialogue d'impression, 2 = Unité du document
valeur	Entier long	→	Valeur de la propriété

Description

La commande QR FIXER PROPRIETE DOCUMENT permet d'afficher la boîte de dialogue d'impression ou de définir l'unité du document présent dans la zone.

Vous pouvez passer dans le paramètre propriété une des constantes du thème QR Propriétés de document :

Constante	Valeur
qr dialogue d'impression	1
qr unité	2

• Si propriété vaut 1, la commande permet de paramétrer l'affichage de la boîte de dialogue d'impression. Dans ce cas :
- Si valeur vaut 0, la boîte de dialogue d'impression n'est pas affichée avant l'impression.
- Si valeur vaut 1, la boîte de dialogue d'impression est affichée avant l'impression (valeur par défaut).

• Si propriété vaut 2, la commande permet de paramétrer l'unité du document. Dans ce cas :
- Si valeur vaut 0, l'unité du document est le point.
- Si valeur vaut 1, l'unité du document est le centimètre.
- Si valeur vaut 2, l'unité du document est le pouce.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si la valeur du paramètre propriété est incorrecte, l'erreur -9852 est générée.

Référence

QR Lire propriete document.

QR FIXER PROPRIETE TEXTE (zone; numColonne; numLigne; propriété; valeur)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne
numLigne	Entier long	→	Numéro de ligne
propriété	Entier long	→	Numéro de propriété
valeur	Entier long	→	Valeur de la propriété définie

Description

La commande QR FIXER PROPRIETE TEXTE permet de définir les propriétés de texte de la cellule désignée par les paramètres numColonne et numLigne.

Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de colonne de la cellule.

Passez dans numLigne la référence de la ligne de la cellule :
• si numLigne vaut -1, elle désigne l'intitulé de la colonne.
• si numLigne vaut -2, elle désigne la zone Détail.
• si numLigne vaut -3, elle désigne la zone Total général.
• si numLigne vaut -4, elle désigne l'en-tête de page.
• si numLigne vaut -5, elle désigne le pied de page.
Vous pouvez passer dans numLigne une des constantes du thème QR Lignes pour Propriétés.
Note : Vous devez passer une valeur dans numColonne même lorsque vous passez -4 ou -5 dans le paramètre numLigne (dans ce cas la valeur de numColonne est inutilisée).
• si numLigne contient une valeur positive, elle désigne la ligne de sous-total correspondante.

Note : Dans les tableaux croisés, le principe est similaire sauf pour les valeurs des lignes, qui sont toujours positives.

Passez dans propriété la valeur de la propriété de texte à modifier. Vous pouvez utiliser les constantes du thème QR Propriétés de texte et les valeurs suivantes peuvent être définies :

Constante (valeur)	Valeur
qr police (1)	Numéro de police retourné par la commande LISTE DES POLICES
qr taille police (2)	Taille de police en points (9 à 255)
qr gras (3)	Attribut gras (0 ou 1)
qr italique (4)	Attribut italique (0 ou 1)
qr souligné (5)	Attribut souligné (0 ou 1)
qr couleur de texte (6)	Numéro de couleur (Entier long)
qr justification (7)	Attribut de justification (0 = par défaut, 1 = gauche, 2 = centre et 3 = droite)
qr couleur de fond (8)	Numéro de couleur de fond
qr couleur de fond alternée (9)	Numéro de couleur de fond alternée

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.
Si le paramètre numLigne est incorrect, l'erreur -9853 est générée.
Si le paramètre propriété est incorrect, l'erreur -9854 est générée.

Exemple

Cette méthode définit plusieurs attributs pour l'intitulé de la première colonne :

```
   `Affecte la police Times:
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr police;Numero de police("Times"))
   `Affecte la taille de police 10 points:
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr taille police;10)
   `Affecte l'attribut gras :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr gras;1)
   `Affecte l'attribut italique :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr italique;1)
   `Affecte l'attribut souligné :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr souligné;1)
   `Affecte la couleur vert clair :
QR FIXER PROPRIETE TEXTE(qr_zone;1;-1;qr couleur de texte;0x0000FF00)
```

Référence

QR Lire propriete texte.

QR FIXER PROPRIETE ZONE

QR FIXER PROPRIETE ZONE (zone; propriété; valeur)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
propriété	Entier long	→	Elément d'interface
valeur	Entier long	→	1 = affiché, 0 = caché

Description

La commande QR FIXER PROPRIETE ZONE vous permet d'afficher ou de masquer dans la zone l'élément d'interface (barre d'outils ou barre de menus) dont la référence est passée dans le paramètre propriété.

La barre de menus et les barres d'outils sont numérotées de 1 à 6 (de haut en bas) et la valeur 7 est associée au menu contextuel. Vous pouvez utiliser les constantes du thème QR Propriétés de zone afin de désigner l'élément d'interface :

Constante (valeur)	Description
qr barre menu (1)	Affichage de la barre de menus (Affichée=1, Cachée=0)
qr barre outils standard (2)	Affichage de la barre d'outils standard (Affichée=1, Cachée=0)
qr barre outils style (3)	Affichage de la barre d'outils Style (Affichée=1, Cachée=0)
qr barre outils opérateurs (4)	Affichage de la barre d'outils Opérateurs (Affichée=1, Cachée=0)
qr barre outils couleur (5)	Affichage de la barre d'outils Couleurs de fond (Affichée=1, Cachée=0)
qr barre outils colonnes (6)	Affichage de la barre d'outils Colonnes (Affichée=1, Cachée=0)
qr menus contextuels (7)	Affichage des menus contextuels (Affichés=1, Cachés=0)

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre propriété est incorrect, l'erreur -9852 est générée.

Référence

QR Lire propriete zone.

QR FIXER SELECTION

QR FIXER SELECTION (zone; gauche; haut; droite; bas)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
gauche	Entier long	→	Limite gauche
haut	Entier long	→	Limite supérieure
droite	Entier long	→	Limite droite
bas	Entier long	→	Limite inférieure

Description

La commande QR FIXER SELECTION permet de sélectionner une cellule, une ligne, une colonne ou encore la totalité de la zone, comme vous le feriez à l'aide de la souris. Cette commande permet également de désélectionner la sélection courante.

gauche contient le numéro de la colonne représentant la limite gauche de la sélection. Si gauche vaut 0, la ligne entière est sélectionnée.
haut contient le numéro de la ligne représentant la limite supérieure de la sélection. Si haut vaut 0, la colonne entière est sélectionnée.
droite contient le numéro de la colonne représentant la limite droite de la sélection.
bas contient le numéro de la ligne représentant la limite inférieure de la sélection.

Notes :

• Si les paramètres gauche and haut valent 0, la totalité de la zone est sélectionnée.
• Pour tout désélectionner, passez -1 dans les paramètres gauche, haut, droite et bas.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR LIRE SELECTION.

QR FIXER TABLE ETAT

QR FIXER TABLE ETAT (zone; table)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
table	Entier long	→	Numéro de table

Description

La commande QR FIXER TABLE ETAT désigne via le paramètre table le numéro de la table courante de l'état rapide dont la référence est passée dans le paramètre zone.

Il est impératif qu'une table soit associée à un état car l'éditeur d'états utilisera la sélection courante de cette table pour afficher les données, effectuer les calculs et propager les liens si nécessaire.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre table est incorrect, l'erreur -9852 est générée.

Référence

QR Lire table etat.

QR FIXER TRIS (zone; tabColonnes{; tabTris})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
tabColonnes	Tableau Réel	→	Colonnes
tabTris	Tableau Réel	→	Ordres de tris

Description

La commande QR FIXER TRIS vous permet de définir l'ordre de tri de chaque colonne de l'état rapide dont la référence est passée dans zone.

tabColonnes : vous devez stocker dans ce tableau le numéro de chaque colonne pour laquelle vous souhaitez définir un ordre de tri.

tabTris : chaque élément de ce tableau doit contenir l'ordre de tri pour la colonne correspondante référencée dans le tableau tabColonnes.
• Si tabTris{$i} vaut 1, le tri est croissant.
• Si tabTris{$i} vaut - 1, le tri est décroissant.

Etats tableaux croisés

Avec ce type d'état, le tableau ne peut pas comporter plus de deux éléments. Vous pouvez uniquement trier les colonnes (1) et les lignes (2). Les données (situées à l'intersection des colonnes et des lignes) ne peuvent pas être triées via cette commande.

Voici le code permettant de trier les lignes uniquement dans un état tableau croisé :

```
TABLEAU REEL($tabColonnes;1)
$tabColonnes{1}:=2
TABLEAU REEL($tabTris;1)
$tabTris{1}:=-1 `Tri décroissant des lignes
QR FIXER TRIS (qr_zone;$tabColonnes;$tabTris)
```

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR LIRE TRIS.

QR FIXER TYPE ETAT (zone; type)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
type	Entier long	→	Type d'état

Description

La commande QR FIXER TYPE ETAT permet de définir le type de l'état rapide présent dans la zone.

Vous pouvez passer dans le paramètre type une des constantes du thème QR Types d'états :

Constante	Valeur
qr état en liste	1
qr état tableau croisé	2

Si vous définissez à l'aide de cette commande un nouveau type pour un état existant, les paramétrages précédents sont supprimés et un nouvel état vide est créé.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si la valeur de type est incorrecte, l'erreur -9852 est générée.

Référence

QR Lire type etat.

QR INSERER COLONNE

QR INSERER COLONNE (zone; numColonne; objet)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne
objet	Champ | Variable | Pointeur	→	Objet à insérer dans la colonne

Description

La commande QR INSERER COLONNE insère ou crée dans zone une colonne à un emplacement spécifique. Les colonnes situées à droite de la colonne ajoutée seront décalées en conséquences.

numColonne indique le numéro de la colonne, correspondant à la position de la colonne — les colonnes sont numérotées de gauche à droite.

La valeur passée dans objet sera l'intitulé par défaut de la colonne.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Exemple

La ligne suivante insère (ou crée) une première colonne dans la zone MaZone et la remplit avec le contenu du champ Noms. L'intitulé par défaut de la colonne sera "Noms" :

```
QR INSERER COLONNE (MaZone;1;->[Table 1]Noms)
```

Référence

QR SUPPRIMER COLONNE.

QR Lire colonne deposee

QR Lire colonne deposee (zone) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
Résultat	Entier long	←	Emplacement du "déposer"

Description

La commande QR Lire colonne deposee retourne une valeur indiquant l'emplacement auquel un “déposer” a été effectué dans zone :

• Si la valeur est négative, elle indique un numéro de colonne (par exemple, -3 indique qu'un “déposer” a été effectué sur la colonne n° 3).

• Si la valeur est positive, elle indique que le “déposer” a été effectué sur le séparateur situé devant la colonne (par exemple, 3 indique qu'un “déposer” a été effectué après la colonne n° 2). Gardez à l'esprit qu'un “déposer” ne peut pas être effectué devant une colonne existante.

Si un numéro de zone invalide est passé, l’erreur -9850 est générée.

Référence

QR SUPPRIMER COLONNE.

QR LIRE DESTINATION

QR LIRE DESTINATION (zone; type{; spécificités})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
type	Entier long	←	Type d'état
spécificités	Chaîne | Variable	←	Spécificités de la destination

Description

La commande QR LIRE DESTINATION retourne le type de destination de l'état rapide contenu dans la zone.

Vous pouvez comparer la valeur obtenue dans le paramètre type avec les constantes du thème QR Destination de sortie.

Le tableau suivant liste les valeurs qui peuvent être retournées dans les paramètres type et spécificités :

Destination	type (valeur)	spécificités
Imprimante	qr imprimante (1)	N.A.
Fichier texte	qr fichier texte (2)	Chemin d'accès du fichier
4D View	qr zone 4D View (3)	N.A.
4D Chart	qr zone 4D Chart (4)	N.A.
Fichier HTML	qr fichier HTML (5)	Chemin d'accès du fichier

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR FIXER DESTINATION.

QR LIRE DONNEES TOTAUX (zone; numColonne; numLigne; opérateur; texte)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne
numLigne	Entier long	→	Numéro de rupture
opérateur	Entier long	←	Opérateur de la cellule
texte	Alpha	←	Contenu de la cellule

Description

Etat en liste
La commande QR LIRE DONNEES TOTAUX permet de récupérer le contenu d'une ligne de rupture spécifique (sous-total ou total général).

Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de colonne de la cellule que vous souhaitez lire.
Passez dans numRupture le numéro de la ligne de rupture à lire (sous-total ou total général). Pour une ligne de sous-total, numRupture correspond au numéro de la ligne. Pour le total général, numRupture vaut -3 (vous pouvez également utiliser la constante qr total général du thème QR Lignes pour Propriétés).

Le paramètre opérateur retourne la valeur cumulée de tous les opérateurs éventuellement présents dans la cellule. Vous pouvez utiliser les constantes du thème QR Opérateurs pour traiter les valeurs retournées :

Constante	Valeur
qr somme	1
qr moyenne	2
qr min	4
qr max	8
qr nombre	16
qr écart type	32

Si opérateur retourne 0, la cellule ne contient aucun opérateur.

texte retourne le texte de la cellule.

Note : Les paramètres opérateur et texte sont mutuellement exclusifs ; en fonction du contenu de la cellule, seul l'un des deux paramètres retournera une valeur.

Etat tableau croisé
La commande QR LIRE DONNEES TOTAUX vous permet de recupérer le contenu d'une cellule spécifique.

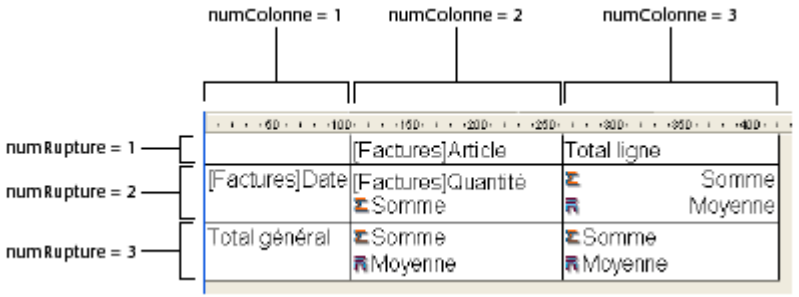
Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de colonne et dans numRupture le numéro de ligne de la cellule que vous souhaitez lire.

Le paramètre opérateur retourne la valeur cumulée de tous les opérateurs éventuellement présents dans la cellule. Utilisez les constantes du thème QR Opérateurs pour évaluer la valeur récupérée (cf. paragraphe précédent).

Le paramètre texte retourne le contenu de la cellule.

L'illustration suivante précise la manière dont les paramètres numColonne et numRupture sont combinés dans un tableau croisé :



Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.
Si le paramètre numRupture est incorrect, l'erreur -9853 est générée.

Référence

QR FIXER DONNEES TOTAUX.

QR LIRE ENCADREMENTS

QR LIRE ENCADREMENTS (zone; colonne; ligne; encadrement; épaisseur{; couleur})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
colonne	Entier long	→	Numéro de colonne
ligne	Entier long	→	Numéro de ligne
encadrement	Entier long	→	Valeur d'encadrement
épaisseur	Entier long	←	Epaisseur de trait
couleur	Entier long	←	Couleur de l'encadrement

Description

La commande QR LIRE ENCADREMENTS retourne les attributs d'encadrement d'une cellule spécifique de zone.

Passez dans zone la référence de la zone d'état rapide.

Passez dans colonne le numéro de colonne de la cellule à lire.

Le paramètre ligne contient le numéro de ligne de la cellule à lire :
• passez -1 pour désigner l'intitulé de l'état,
• passez -2 pour désigner la zone Détail de l'état,
• passez -3 pour désigner la zone Total général,
• passez une valeur entière positive pour désigner la ligne de sous-total correspondante.
Vous pouvez également utiliser les constantes correspondantes dans le thème QR Lignes pour Propriétés.

Le paramètre encadrement permet d'indiquer la bordure de cellule à lire :
• 1 désigne la bordure gauche,
• 2 désigne la bordure supérieure,
• 4 désigne la bordure droite,
• 8 désigne la bordure inférieure,
• 16 désigne la bordure intérieure verticale,
• 32 désigne la bordure intérieure horizontale.

Vous pouvez également utiliser les constantes correspondantes dans le thème QR Encadrements.

Note : A la différence de la commande QR FIXER ENCADREMENTS, QR LIRE ENCADREMENTS n'accepte pas de valeurs cumulées. Vous devez tester séparément toutes les valeurs pour obtenir une description globale de l'encadrement de la cellule.

Le paramètre épaisseur retourne l'épaisseur de l'encadrement :
• 0 indique une épaisseur nulle (pas de bordure),
• 1 indique une épaisseur d'1/4 point,
• 2 indique une épaisseur d'1/2 point,
• 3 indique une épaisseur d'1 point,
• 4 indique une épaisseur de 2 points.

Le paramètre couleur retourne le numéro de la couleur de la bordure.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre colonne est incorrect, l'erreur -9852 est générée.
Si le paramètre ligne est incorrect, l'erreur -9853 est générée.
Si le paramètre encadrement est incorrect, l'erreur -9854 est générée.

Référence

QR FIXER ENCADREMENTS.

QR LIRE ENTETE ET PIED DE PAGE (zone; sélecteur; titreGauche; titreCentre; titreDroit; hauteur{; image{; alignementImage}})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
sélecteur	Entier long	→	1 = En-tête, 2 = Pied de page
titreGauche	Alpha	←	Texte affiché sur le côté gauche
titreCentre	Alpha	←	Texte affiché au centre
titreDroit	Alpha	←	Texte affiché sur le côté droit
hauteur	Numérique	←	Hauteur de l'en-tête ou du pied de page
image	Image	←	Image à afficher
alignementImage	Entier long	←	Alignement de l'image

Description

La commande QR LIRE ENTETE ET PIED DE PAGE vous permet de récupérer le contenu et la taille de l'en-tête et du pied de page de la zone.

Le paramètre sélecteur vous permet de définir la zone à lire :
• si sélecteur vaut 1, les informations de l''en-tête seront récupérées ;
• si sélecteur vaut 2, les informations du pied de page seront récupérées.

Les paramètres titreGauche, titreCentre et titreDroite retournent le contenu des en-têtes ou pieds de page situés respectivement à gauche, au centre et à droite.

Le paramètre hauteur retourne la hauteur de l'en-tête ou du pied de page, exprimée dans l'unité courante de l'état rapide.

Le paramètre image retourne le cas échéant l'image affichée dans l'en-tête ou le pied de page.
Le paramètre alignementImage retourne la propriété d'alignement de l'image :
• si alignementImage vaut 0, l'image est alignée sur la gauche.
• si alignementImage vaut 1, l'image est centrée.
• si alignementImage vaut 2, l'image est alignée sur la droite.

Si un numéro de zone invalide est passé, l’erreur -9850 est générée.
Si le paramètre sélecteur est incorrect, l’erreur -9852 est générée.

Exemple

La méthode suivante affiche le contenu et la hauteur des libellés des en-têtes :

```
QR LIRE ENTETE ET PIED DE PAGE(MaZone;1;$TexteGauche;$TexteCentre;$TexteDroite;
                                                                    $Hauteur)
Au cas ou
    : ($TexteGauche # "")
        ALERTE("Libellé de l'en-tête de gauche : "+Caractere(34)+$TexteGauche+
                                                                    Caractere(34))
    : ($TexteCentre # "")
        ALERTE("Libellé de l'en-tête du centre : "+Caractere(34)+$TexteCentre+
                                                                    Caractere(34))
    : ($TexteDroite # "")
        ALERTE("Libellé de l'en-tête de droite : "+Caractere(34)+$TexteDroite+
                                                                    Caractere(34))
    Sinon
        ALERTE("Aucun libellé d'en-tête dans cet Etat.")
    Fin de cas
ALERTE("Hauteur des en-têtes : "+Chaine($Hauteur))
```

Référence

QR FIXER ENTETE ET PIED DE PAGE.

QR LIRE ESPACEMENT TOTAUX

QR LIRE ESPACEMENT TOTAUX (zone; sousTotal; valeur)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
sousTotal	Entier long	→	Numéro de sous-total
valeur	Entier long	←	0=pas d'espace, 32000=insère une saut de page, >0=espace ajouté en haut du niveau de rupture, <0=augmentation proportionnelle

Description

La commande QR LIRE ESPACEMENT TOTAUX permet de récupérer la valeur de l'espacement ajouté au-dessus d'une ligne de sous-total. Elle ne peut être utilisée qu'avec des états en liste.

Le paramètre zone contient la référence de la zone d'état rapide.
Le paramètre sousTotal désigne le niveau de sous-total (ou de rupture) dont vous souhaitez connaître l'espacement. Ce paramètre contient une valeur comprise entre 1 et le nombre de lignes de sous-total/rupture.

Le paramètre valeur retourne la valeur de l'espacement :
• Si valeur vaut 0, aucun espacement n'est ajouté.
• Si valeur vaut 32000, un saut de page est ajouté.
• Si valeur est une valeur positive, elle exprime l'espacement en pixels.
• Si valeur est une valeur négative, elle exprime l'espacement en pourcentage de la ligne de sous-total. Par exemple, la valeur -100 indique un espace au-dessus de la ligne de sous-total correspondant à 100% de la hauteur courante de la ligne.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre sousTotal est incorrect, l'erreur -9852 est générée.

Référence

QR FIXER ESPACEMENT TOTAUX.

QR LIRE INFO COLONNE (zone; numCol; titre; objet; cachée; taille; valeursRépétées; format)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numCol	Entier long	→	Numéro de colonne
titre	Alpha	←	Titre de la colonne
objet	Champ | Variable	←	Objet affecté à la colonne
cachée	Entier long	←	0 = visible, 1 = invisible
taille	Entier long	←	Largeur de la colonne
valeursRépétées	Entier long	←	0 = non répétées, 1 = répétées
format	Texte	←	Format d'affichage des données

Description

Etats en liste
La commande QR LIRE INFO COLONNE vous permet de récupérer les paramètres d'une colonne existante de l'état présent dans la zone.

Passez dans zone la référence de la zone d'état rapide et dans numColonne le numéro de la colonne à définir.

Le paramètre titre retourne l'intitulé de l'en-tête de la colonne.

Le paramètre objet retourne la référence de l'objet associé à la colonne (variable, champ ou formule).

Le paramètre cachée indique si la colonne est affichée ou masquée :
• si cachée vaut 1, la colonne est masquée ;
• si cachée vaut 0, la colonne est affichée.

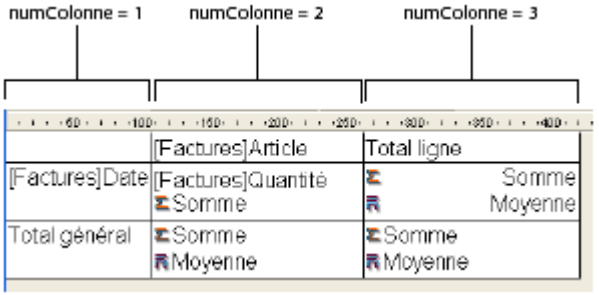
Le paramètre taille retourne la taille en pixels de la colonne. Si la valeur retournée est négative, la taille de la colonne est automatique.

valeursRépétées retourne le statut de la propriété de répétition des données. Par exemple, si la valeur d'un champ ou d'une variable ne change pas d'un enregistrement à l'autre, il est possible de la répéter ou non dans chaque ligne de la colonne.
• si valeursRépétées vaut 0, les valeurs ne sont pas répétées.
• si valeursRépétées vaut 1, les valeurs sont répétées.

Le paramètre format retourne le format d'affichage de la colonne. Vous pouvez utiliser tout format d'affichage standard de 4D compatible avec les données affichées dans la colonne.

Etats tableaux croisés
Avec ce type d'état, la commande QR LIRE INFO COLONNE permet de récupérer globalement les mêmes paramètres que ceux décrits dans le paragraphe ci-dessus, toutefois les zones auxquelles ils s'appliquent sont différentes et varient en fonction du paramètre à lire.
En outre, les paramètres titre, cachée et valeursRépétées ne sont pas utilisés lorsque vous travaillez avec des états tableaux croisés, les valeurs retournées dans ces paramètres ne sont donc pas significatives.
La valeur à passer dans le paramètre numColonne dépend de l'opération que vous souhaitez effectuer : lire la taille de la colonne ou lire la source de données et le format d'affichage.

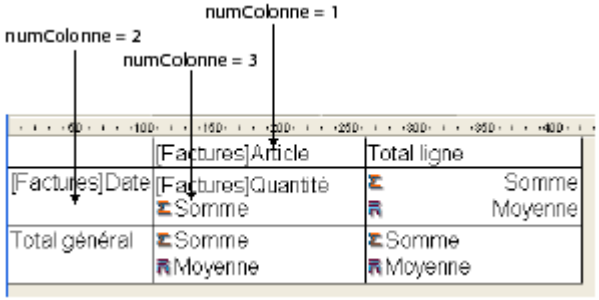
• *Taille de la colonne*
Il s'agit d'un attribut “visuel”, par conséquent les colonnes sont simplement numérotées de gauche à droite, comme illustré ci-dessous :



La méthode suivante affecte une taille automatique à toutes les colonnes d'un état en tableau croisé et laisse les autres éléments inchangés :

```
Boucle ($i;1;3)
    QR LIRE INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;$taille;$rep;$format)
    QR FIXER INFO COLONNE(qr_zone;$i;$titre;$obj;$caché;0;$rep;$format)
Fin de boucle
```

• *Source de données (objet) et format d'affichage*
Dans ce cas, la numérotation des colonnes s'effectue de la manière suivante :



Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.

Référence

QR FIXER INFO COLONNE, QR FIXER INFO LIGNE, QR Lire info ligne.

QR Lire info ligne

QR Lire info ligne (zone; ligne) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
ligne	Entier long	→	Ligne
Résultat	Entier long	←	0 = Visible, 1 = Cachée

Description

La commande QR Lire info ligne indique si la ligne désignée par le paramètre ligne est affichée ou masquée dans la zone.

Le paramètre ligne désigne la ligne à vérifier :
• passez une valeur entière positive pour désigner la ligne de sous-total correspondante,
• passez -1 pour désigner l'intitulé de l'état,
• passez -2 pour désigner la zone Détail de l'état,
• passez -3 pour désigner la zone Total général.
Vous pouvez également utiliser l'une des trois constantes correspondantes dans le thème QR Lignes pour Propriétés.

La valeur retournée par QR Lire info ligne indique si le contenu de la ligne est affiché ou masqué. Si la fonction retourne 1, le contenu de la ligne est masqué ; si elle retourne 0, le contenu de la ligne est affiché.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre ligne est incorrect, l'erreur -9852 est générée.

Référence

QR FIXER INFO COLONNE, QR FIXER INFO LIGNE, QR LIRE INFO COLONNE.

QR Lire modele HTML

QR Lire modele HTML (zone) → Texte

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
Résultat	Texte	←	Code HTML utilisé comme modèle

Description

La commande QR Lire modele HTML retourne le modèle HTML utilisé pour la zone d'état rapide référencée par zone. La valeur retournée, de type texte, contient la totalité du code HTML utilisé comme modèle.

Si aucun modèle spécifique n'a été défini, le code du modèle par défaut est retourné.
A noter que si le format de destination HTML n'a pas été défini pour l'état (manuellement ou par programmation), aucune valeur n'est retournée.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR FIXER MODELE HTML.

QR Lire propriete document

QR Lire propriete document (zone; propriété) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
propriété	Entier long	→	1=Dialogue d'impression, 2=Unité du document
Résultat	Entier long	←	Valeur de la propriété

Description

La commande QR Lire propriete document vous permet de connaître la valeur courante de la propriété d'affichage de la boîte de dialogue d'impression ou de l'unité du document présent dans la zone.

Vous pouvez passer dans le paramètre propriété une des constantes du thème QR Propriétés de document :

Constante	Valeur
qr dialogue d'impression	1
qr unité	2

• Si propriété vaut 1, la commande retourne le paramétrage de l'affichage de la boîte de dialogue d'impression. Dans ce cas :
- Si la commande retourne 0, la boîte de dialogue d'impression n'est pas affichée avant l'impression.
- Si la commande retourne 1, la boîte de dialogue d'impression est affichée avant l'impression (valeur par défaut).

• Si propriété vaut 2, la commande retourne le paramétrage de l'unité du document. Dans ce cas :
- Si la commande retourne 0, l'unité du document est le point.
- Si la commande retourne 1, l'unité du document est le centimètre.
- Si la commande retourne 2, l'unité du document est le pouce.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si la valeur du paramètre propriété est incorrecte, l'erreur -9852 est générée.

Référence

QR FIXER PROPRIETE DOCUMENT.

QR Lire propriete texte

QR Lire propriete texte (zone; numColonne; numLigne; propriété) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne
numLigne	Entier long	→	Numéro de rupture
propriété	Entier long	→	Numéro de propriété
Résultat	Entier long	←	Valeur de la propriété

Description

La commande QR Lire propriete texte retourne la valeur courante de la propriété de texte dans la cellule de zone désignée par numColonne et numLigne.

Passez dans zone la référence de la zone d'état rapide.

Passez dans numColonne le numéro de colonne de la cellule.

Passez dans numLigne la référence de la ligne de la cellule :
• si numLigne vaut -1, elle désigne l'intitulé de la colonne.
• si numLigne vaut -2, elle désigne la zone Détail.
• si numLigne vaut -3, elle désigne la zone Total général.
• si numLigne vaut -4, elle désigne l'en-tête de page.
• si numLigne vaut -5, elle désigne le pied de page.
Vous pouvez passer dans numLigne une des constantes du thème QR Lignes pour Propriétés.
Note : Vous devez passer une valeur dans numColonne même lorsque vous passez -4 ou -5 dans le paramètre numLigne (dans ce cas la valeur de numColonne est inutilisée).
• si numLigne contient une valeur positive, elle désigne la ligne de sous-total correspondante.

Note : Dans les tableaux croisés, le principe est similaire sauf pour les valeurs des lignes, qui sont toujours positives.

Passez dans propriété la valeur de la propriété de texte à lire. Vous pouvez utiliser les constantes du thème QR Propriétés de texte et les valeurs suivantes peuvent être retournées :

Constante (valeur)	Valeurs retournées
qr police (1)	Numéro de police retourné par la commande Numero de police
qr taille police (2)	Taille de police en points (9 à 255)
qr gras (3)	Attribut gras (0 ou 1)
qr italique (4)	Attribut italique (0 ou 1)
qr souligné (5)	Attribut souligné (0 ou 1)
qr couleur de texte (6)	Numéro de couleur (Entier long)
qr justification (7)	Attribut de justification (0 = par défaut, 1 = gauche, 2 = centre et 3 = droite)
qr couleur de fond (8)	Numéro de couleur de fond
qr couleur de fond alternée (9)	Numéro de couleur de fond alternée

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.
Si le paramètre numLigne est incorrect, l'erreur -9853 est générée.
Si le paramètre propriété est incorrect, l'erreur -9854 est générée.

Référence

QR FIXER PROPRIETE TEXTE.

QR Lire propriete zone

QR Lire propriete zone (zone; propriété) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
propriété	Entier long	→	Elément d'interface
Résultat	Entier long	←	1 = affiché, 0 = caché

Description

La commande QR Lire propriete zone retourne 0 si l'élément d'interface (barre d'outils ou barre de menus) dont la référence est passée dans le paramètre propriété est masqué dans la zone, sinon elle retourne 1.

La barre de menus et les barres d'outils sont numérotées de 1 à 6 (de haut en bas) et la valeur 7 est associée au menu contextuel. Vous pouvez utiliser les constantes du thème QR Propriétés de zone afin de désigner l'élément d'interface :

Constante (valeur)	Description
qr barre menu (1)	Affichage de la barre de menus (Affichée=1, Cachée=0)
qr barre outils standard (2)	Affichage de la barre d'outils standard (Affichée=1, Cachée=0)
qr barre outils style (3)	Affichage de la barre d'outils Style (Affichée=1, Cachée=0)
qr barre outils opérateurs (4)	Affichage de la barre d'outils Opérateurs (Affichée=1, Cachée=0)
qr barre outils couleur (5)	Affichage de la barre d'outils Couleurs de fond (Affichée=1, Cachée=0)
qr barre outils colonnes (6)	Affichage de la barre d'outils Colonnes (Affichée=1, Cachée=0)
qr menus contextuels (7)	Affichage des menus contextuels (Affichés=1, Cachés=0)

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre propriété est incorrect, l'erreur -9852 est générée.

Référence

QR FIXER PROPRIETE ZONE.

QR LIRE SELECTION

QR LIRE SELECTION (zone; gauche; haut{; droite{; bas}})

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
gauche	Entier long	←	Limite gauche
haut	Entier long	←	Limite supérieure
droite	Entier long	←	Limite droite
bas	Entier long	←	Limite inférieure

Description

La commande QR LIRE SELECTION retourne les coordonnées de la sélection courante de la zone.

gauche retourne le numéro de la colonne représentant la limite gauche de la sélection. Si gauche vaut 0, la ligne entière est sélectionnée.
haut retourne le numéro de la ligne représentant la limite supérieure de la sélection. Si haut vaut 0, la colonne entière est sélectionnée.

Note : Si les paramètres gauche and haut valent 0, la totalité de la zone est sélectionnée.

droite retourne le numéro de la colonne représentant la limite droite de la sélection.
bas retourne le numéro de la ligne représentant la limite inférieure de la sélection.

Note : Si la zone ne contient aucune sélection, les paramètres gauche, haut, droite et bas retournent -1.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR FIXER SELECTION.

QR Lire statut commande

QR Lire statut commande (zone; numCommande{; valeur}) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numCommande	Entier long	→	Numéro de commande
valeur	Texte | Entier long	←	Valeur du sous-élément sélectionné
Résultat	Entier long	←	Statut de la commande

Description

La commande QR Lire statut commande retourne 0 si la commande désignée par le paramètre numCommande est inactivée et 1 si elle est activée.

valeur retourne la valeur du sous-élément sélectionné, le cas échéant. Par exemple, si la commande sélectionnée est la liste déroulante des polices (1000) et que la police choisie est l'Arial, valeur vaut “Arial” ; si la commande sélectionnée est le menu des couleurs (1002, 1003 ou 1004), valeur retourne le numéro de la couleur.

La commande QR Lire statut commande peut être utilisée dans deux types de contextes :
• comme simple instruction pour déterminer si une commande est active ou non.
• dans une méthode installée par QR APPELER SUR COMMANDE afin de connaître le sous-élément sélectionné. Dans cette méthode, $1 contient la référence de la zone et $2 le numéro de la commande.

Passez dans paramètre numCommande une des constantes du thème QR Commandes.

Si un numéro de zone invalide est passé, l’erreur -9850 est générée.
Si le paramètre numCommande est incorrect, l’erreur -9852 est générée.

Référence

QR APPELER SUR COMMANDE, QR EXECUTER COMMANDE.

QR Lire table etat

QR Lire table etat (zone) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
Résultat	Entier long	←	Numéro de table

Description

La commande QR Lire table etat retourne le numéro de la table courante de l'état désigné par le paramètre zone.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR FIXER TABLE ETAT.

QR LIRE TRIS (zone; tabColonnes; tabTris)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
tabColonnes	Tableau Réel	←	Colonnes triées
tabTris	Tableau Réel	←	Ordres de tris

Description

La commande QR LIRE TRIS remplit deux tableaux réels :

• tabColonnes
Ce tableau contient toutes les colonnes auxquelles un ordre de tri a été associé.

• tabTris
Chaque élément de ce tableau fournit l'ordre de tri courant de la colonne correspondante.
- si tabTris{$i} vaut 1, le tri est croissant.
- si tabTris{$i} vaut - 1, le tri est décroissant.

Etats tableaux croisés

Avec ce type d'état, les tableaux ne peut pas comporter plus de deux éléments puisque les tris ne peuvent être effectués que sur les colonnes (1) et les lignes (2) (valeurs pour tabColonnes).

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR FIXER TRIS.

QR Lire type etat

QR Lire type etat (zone) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
Résultat	Entier long	←	Type d'état

Description

La commande QR Lire type etat retourne le type d'état présent dans la zone.

- Si la commande retourne 1, l'état est de type liste
- Si la commande retourne 2, l'état est de type tableau croisé

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR FIXER TYPE ETAT.

QR Nombre de colonnes

QR Nombre de colonnes (zone) → Entier long

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
Résultat	Entier long	←	Nombre de colonnes dans la zone

Description

La commande QR Nombre de colonnes retourne le nombre de colonnes présentes dans l'état rapide désigné par le paramètre zone.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Exemple

L'exemple suivant permet d'ajouter une colonne supplémentaire à droite de la dernière colonne de la zone :

```
$NbCol:=QR Nombre de colonnes(MaZone)
QR INSERER COLONNE(MaZone;$NbCol+1;->[Table 1]Noms)
```

Référence

QR INSERER COLONNE, QR SUPPRIMER COLONNE.

QR SUPPRIMER COLONNE

QR SUPPRIMER COLONNE (zone; numColonne)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone
numColonne	Entier long	→	Numéro de colonne

Description

La commande QR SUPPRIMER COLONNE supprime de la zone la colonne dont le numéro a été passé dans numColonne. Cette commande ne peut pas être utilisée avec les états en tableau croisé.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.
Si le paramètre numColonne est incorrect, l'erreur -9852 est générée.

Exemple

Cet exemple supprime la troisième colonne de l'état :

```
Si(QR Lire type etat(MaZone )=qr état en liste)
    QR SUPPRIMER COLONNE (MaZone;3)
Fin de si
```

Référence

QR INSERER COLONNE.

QR SUPPRIMER ZONE HORS ECRAN (zone)

Paramètre	Type		Description
zone	Entier long	→	Référence de la zone

Description

La commande QR SUPPRIMER ZONE HORS ECRAN efface de la mémoire la zone hors écran dont la référence a été passée dans le paramètre zone.

Si un numéro de zone invalide est passé, l'erreur -9850 est générée.

Référence

QR Creer zone hors ecran.

19

Evénements formulaire

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne l'événement Sur activation.

Activation → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai si le cycle d'exécution est en activation

Description

Activation retourne Vrai dans une méthode formulaire lorsque la fenêtre contenant le formulaire passe au premier plan.

ATTENTION : N'appelez pas de commandes telles que TRACE ou ALERTE dans la phase Activation d'un formulaire, car cela provoquerait une boucle sans fin.

Note : Si vous voulez que le cycle d'exécution Activation soit généré, assurez-vous que la propriété d'événement Sur activation du formulaire et/ou des objet(s) est sélectionnée en mode Développement. Cette propriété est automatiquement définie pour les bases converties.

Référence

Desactivation, Evenement formulaire.

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne un événement tel que Sur appel exterieur.

Appel exterieur → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description
Si vous voulez que le cycle d'exécution Appel exterieur soit généré, vérifiez que la propriété d'événement Sur appel exterieur du formulaire et/ou des objets est sélectionnée en mode Développement.

Référence
APPELER PROCESS, Evenement formulaire.

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne un événement tel que Sur validation.

Apres → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Si vous souhaitez que la phase **Apres** du cycle d'exécution soit générée, assurez-vous que l'événement Sur validation a bien été sélectionné, en mode Développement, dans les propriétés du formulaire et/ou des objets concernés.

Référence

Evenement formulaire.

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne l'événement Sur chargement.

Avant → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Si vous souhaitez que la phase **Avant** du cycle d'exécution soit générée, assurez-vous que l'événement Sur chargement a bien été sélectionné, en mode Développement, dans les propriétés du formulaire et/ou des objets concernés.

Référence

Evenement formulaire.

Clic contextuel → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai si un clic contextuel a été détecté, sinon Faux

Description

La commande Clic contextuel retourne Vrai si un clic de type contextuel a été effectué :
• Sous Windows et Mac OS, les clics contextuels sont effectués avec le bouton droit de la souris.
• Sous Mac OS, des clics contexuels peuvent également être générés à l'aide de la combinaison **Control+clic**.

Cette commande doit être appelée uniquement dans le cadre de l'événement formulaire Sur clic. Il est donc nécessaire de vérifier en mode Développement que cet événement a bien été coché dans les Propriétés du formulaire et/ou de l'objet.

Exemple

Cette méthode, associée à une zone de défilement, permet de changer la valeur d'un élément de tableau à l'aide d'un menu contextuel :

```
Si(Clic contextuel)
    Si (Pop up menu("Vrai;Faux")=1)
        monTableau{monTableau}:="Vrai"
    Sinon
        monTableau{monTableau}:="Faux"
    Fin de si
Fin de si
```

Référence

Clic droit, Evenement formulaire.

Clic droit → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai si un clic droit a été détecté, sinon Faux

Description

La commande Clic droit retourne Vrai si un clic effectué avec le bouton droit de la souris a été effectué.

Cette commande doit être appelée uniquement dans le cadre de l'événement formulaire Sur clic. Il est donc nécessaire de vérifier en mode Développement que cet événement a bien été coché dans les Propriétés du formulaire et/ou de l'objet.

Référence

Clic contextuel, Evenement formulaire.

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne l'événement Sur désactivation.

Desactivation → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai si le cycle d'exécution est en désactivation

Description

Desactivation retourne Vrai dans une méthode formulaire ou méthode objet lorsque la fenêtre appartenant au process du premier plan, contenant le formulaire, passe à l'arrière-plan.

Si vous voulez que le cycle d'exécution Desactivation soit généré, vérifiez que la propriété d'événement Sur désactivation du formulaire et/ou des objets est sélectionnée en mode Développement.

Référence

Activation, Evenement formulaire.

version 5

Note de compatibilité
Cette commande a été conservée pour des raisons de compatibilité. A partir de la version 6 de 4D, il est préférable d'utiliser la commande Evenement formulaire et de tester si elle retourne l'événement Sur entête.

En entete → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Si vous souhaitez que le cycle d'exécution En entete soit généré, assurez-vous que l'événement formulaire Sur entête a bien été activé dans la boîte de dialogue des propriétés du formulaire ou du (des) objet(s) sélectionné(s), en mode Développement.

Référence

En pied, En rupture, Pendant.

En pied

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne l'événement Sur impression pied de page.

En pied → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description
Si vous voulez que le cycle d'exécution En pied soit généré, vérifiez que la propriété d'événement Sur impression pied de page du formulaire et/ou des objets est sélectionnée en mode Développement.

Référence

En entete, En rupture, Pendant.

version 5

Note de compatibilité
Cette commande a été conservée pour des raisons de compatibilité. A partir de la version 6 de 4D, il est préférable d'utiliser la commande Evenement formulaire et de tester si elle retourne l'événement Sur impression sous total.

En rupture → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description
Si vous souhaitez que le cycle d'exécution En rupture soit généré, assurez-vous que l'événement formulaire Sur impression sous total a bien été activé dans la boîte de dialogue des propriétés du formulaire ou du (des) objet(s) sélectionné(s), en mode Développement.

Référence

En entete, En pied, Pendant.

Evenement formulaire → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Numéro d'événement formulaire

Description

Evenement formulaire retourne une valeur numérique qui identifie le type d'événement formulaire qui vient de se produire. Généralement, Evenement formulaire s'utilise dans une méthode formulaire ou une méthode objet.

4D fournit des constantes prédéfinies (placées dans le thème “Evénements formulaire”) permettant de comparer les valeurs retournées par la commande Evenement formulaire.

Certains événements sont génériques (générés pour tout type d'objet), d'autres sont spécifiques à un type d'objet particulier.

Evénements génériques
Les événements suivants sont générés pour tout formulaire ou objet :

Constante	Valeur	Description
Sur chargement	1	Le formulaire s'affiche ou s'imprime
Sur libération	24	Le formulaire se referme et est déchargé
Sur validation	3	La saisie des données dans l'enregistrement est validée
Sur clic	4	Un clic est survenu sur un objet
Sur double clic	13	Un double-clic est survenu sur un objet
Sur avant frappe clavier	17	Un caractère vient d'être saisi dans l'objet qui a le focus Lire texte edite retourne le contenu sans ce caractère
Sur après frappe clavier	28	Un caractère vient d'être saisi dans l'objet qui a le focus Lire texte edite retourne le contenu avec ce caractère
Sur après modification	45	Le contenu de l'objet saisissable qui a le focus vient d'être modifié
Sur gain focus	15	Un objet de formulaire prend le focus
Sur perte focus	14	Un objet de formulaire perd le focus
Sur activation	11	La fenêtre du formulaire passe au premier plan
Sur désactivation	12	La fenêtre du formulaire passe en arrière-plan

Sur appel extérieur	10	Le formulaire a reçu un appel de la commande APPELER PROCESS
Sur déposer	16	Des données sont déposées sur un objet
Sur glisser	21	Des données peuvent être déposées sur un objet
Sur début glisser	46	Un objet est en cours de glisser
Sur début survol	35	Le curseur de la souris entre dans la zone graphique d'un objet
Sur survol	37	Le curseur de la souris bouge (d'au moins un pixel) alors qu'il se trouve dans la zone graphique d'un objet
Sur fin survol	36	Le curseur de la souris sort de la zone graphique d'un objet
Sur menu sélectionné	18	Une commande de menu a été sélectionnée
Sur données modifiées	20	Les données d'un objet ont été modifiées
Sur appel zone du plug in	19	Un plug-in demande que sa méthode objet soit exécutée
Sur entête	5	L'en-tête du formulaire va être imprimé ou affiché
Sur impression corps	23	Le corps du formulaire va être imprimé
Sur impression sous total	6	Une rupture du formulaire va être imprimée
Sur impression pied de page	7	Le pied de page du formulaire va être imprimé
Sur case de fermeture	22	On a cliqué sur la case de fermeture de la fenêtre
Sur affichage corps	8	Un enregistrement va être affiché dans la liste
Sur ouverture corps	25	On a double-cliqué sur un enregistrement et on passe au formulaire entrée
Sur fermeture corps	26	Le formulaire entrée se referme et on retourne au formulaire sortie
Sur nouvelle sélection	31	• List box : la sélection courante de lignes ou de colonnes est modifiée • Enregistrements en liste : l'enregistrement courant ou la sélection courante de lignes est modifié(e) dans un formulaire en liste ou un sous-formulaire • Liste hiérarchique : la sélection dans la liste est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier
Sur chargement ligne	40	En mode saisie en liste, un enregistrement est chargé en modification (l'utilisateur a cliqué sur la ligne de l'enregistrement et un champ passe en édition)
Sur minuteur	27	Le nombre de ticks défini par FIXER MINUTEUR est atteint
Sur redimensionnement	29	La fenêtre du formulaire est redimensionnée

List box
Les événements suivants sont générés uniquement pour les objets de type List box :

Constante	Valeur	Description
Sur avant saisie	41	Une cellule de list box est sur le point de passer en mode édition
Sur déplacement colonne	32	Une colonne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur déplacement ligne	34	Une ligne de list box est déplacée par l'utilisateur via le glisser-déposer
Sur redimensionnement colonne	33	La largeur d'une colonne de list box est modifiée
Sur clic entête	42	Un clic est survenu dans l'en-tête d'une colonne de list box
Sur après tri	30	Un tri standard vient d'être effectué dans une colonne de list box

Boutons 3D
Les événements suivants sont générés uniquement pour les objets de type bouton 3D :

Constante	Valeur	Description
Sur clic long	39	Un bouton 3D reçoit un clic et le bouton de la souris reste enfoncé pendant un certain laps de temps
Sur clic flèche	38	La zone "flèche" d'un bouton 3D reçoit un clic

Listes hiérarchiques
Les événements suivants sont générés uniquement pour les objets de type Liste hiérarchique :

Constante	Valeur	Description
Sur déployer	43	Un élément de liste hiérarchique a été déployé via un clic ou une touche du clavier
Sur contracter	44	Un élément de liste hiérarchique a été contracté via un clic ou une touche du clavier

Zones Web
Les événements suivants sont générés uniquement pour les objets de type Zone Web :

Constante	Valeur	Description
Sur chargement ressource URL	48	Une nouvelle ressource est chargée dans la zone Web
Sur début chargement URL	47	Un nouvel URL est chargé dans la zone Web
Sur erreur chargement URL	50	Une erreur s'est produite durant le chargement de l'URL
Sur filtrage URL	51	Un URL a été bloqué par la zone Web
Sur fin chargement URL	49	Toutes les ressources de l'URL ont été chargées
Sur ouverture lien externe	52	Un URL externe a été ouvert dans le navigateur
Sur refus ouverture fenêtre	53	Une fenêtre pop up a été bloquée

Note : Les événements spécifiques des formulaires de sortie ne peuvent pas être utilisés avec les **formulaires projet**. Il s'agit de : Sur affichage corps, Sur ouverture corps, Sur fermeture corps, Sur chargement ligne, Sur entête, Sur impression corps, Sur impression sous total, Sur impression pied de page.

Evénements et méthodes

Lorsqu'un événement formulaire se produit, 4D effectue les actions suivantes :

• En premier lieu, il examine chaque objet du formulaire et appelle la méthode de ceux dont la propriété d'événement correspondante a été sélectionnée et qui sont impliqués dans l'événement.

• Ensuite, il appelle la méthode formulaire si la propriété d'événement correspondante a été sélectionnée pour le formulaire.

Les différentes méthodes objet ne sont pas appelées dans un ordre particulier. La règle est que les méthodes objet sont toujours appelées avant la méthode formulaire. Dans le cas des sous-formulaires, les méthodes objet du formulaire sortie du sous-formulaire sont d'abord appelées, puis la méthode formulaire du formulaire sortie, puis enfin 4D appelle les méthodes objet du formulaire parent. Autrement dit, lorsqu'un objet est un sous-formulaire, 4D utilise la même règle pour les méthodes formulaire et objet dans le sous-formulaire.

Lorsque, pour un événement particulier, la propriété d'événement du formulaire n'est pas sélectionnée, cela n'empêche pas les appels aux méthodes des objets pour lesquels l'événement est sélectionné. Autrement dit, la sélection ou la désélection d'un événement au niveau du formulaire n'a pas d'effet sur les propriétés d'événements des objets.

ATTENTION : Ce principe ne s'applique pas aux événements Sur chargement et Sur libération. Ces événements ne seront générés pour un objet que si les propriétés d'événement correspondantes ont été sélectionnées à la fois pour l'objet et pour le formulaire auquel il appartient. Si les propriétés sont sélectionnées pour l'objet uniquement, les événements ne seront pas générés ; ces deux événements doivent être sélectionnés au niveau du formulaire.

Le nombre d'objets impliqués par un événement dépend de la nature de l'événement. En particulier :

• Pour l'événement Sur chargement, les méthodes objet de tous les objets du formulaire (sur toutes les pages) pour lequels la propriété d'événement Sur chargement est sélectionnée seront appelées. Si l'événement Sur chargement est sélectionné pour le formulaire, la méthode formulaire sera appelée.

• Pour les événements Sur activation ou Sur redimensionnement, aucune méthode objet ne sera appelée car ces événements s'appliquent au formulaire, pas à un objet en particulier. Par conséquent, si ces événements sont sélectionnés pour le formulaire, seule la méthode formulaire sera appelée.

• L'événement Sur minuteur n'est généré que si la méthode formulaire contient un appel préalable à la commande FIXER MINUTEUR. Seule la méthode formulaire reçoit cet événement, aucune méthode objet ne sera appelée.

• Pour l'événement Sur glisser, seule la méthode de l'objet déposable impliqué par l'événement sera appelée (si la propriété d'événement "Déposable" est sélectionnée pour l'objet). La méthode formulaire ne sera pas appelée.

• A l'inverse, pour l'événement Sur début glisser, la méthode objet ou la méthode formulaire de l'objet glissé sera appelée (si la propriété d'événement "Glissable" est sélectionnée pour l'objet).

ATTENTION : Contrairement aux autres événements, pendant un événement Sur début glisser ou Sur glisser, la méthode appelée est exécutée dans le contexte du process de l'objet source du glisser-déposer, et non dans celui du process de l'objet de destination. Pour plus d'informations, reportez-vous à la section Présentation du Glisser-Déposer.

• Si les événements Sur début survol, Sur survol et Sur fin survol sont cochés pour le formulaire, ils sont générés pour chaque objet du formulaire. S'ils sont cochés pour un objet, ils sont générés pour cet objet uniquement. En cas de superposition d'objets, l'événement est généré par le premier objet capable de le gérer dans l'ordre des plans du haut vers le bas. Les objets rendus invisibles par la commande CHOIX VISIBLE ne génèrent pas ces événements. Pendant la saisie d'un objet, les autres objets peuvent recevoir les événements de survol en fonction de la position de la souris.

• Enregistrements en liste : l'enchaînement d'appels des méthodes et des événements formulaires dans les formulaires liste affichés via MODIFIER SELECTION / VISUALISER SELECTION et les sous-formulaires est le suivant :

```
Pour chaque objet de la zone d'en-tête :
    Méthode objet avec événement Sur entête
Méthode formulaire avec événement Sur entête
Pour chaque enregistrement :
    Pour chaque objet de la zone de corps :
        Méthode objet avec événement Sur affichage corps
    Méthode formulaire avec événement Sur affichage corps
```

L'appel depuis les événements Sur affichage corps et Sur entête d'une commande 4D provoquant l'affichage d'une boîte de dialogue est interdit et provoque une erreur de syntaxe. Les commandes concernées sont notamment : ALERTE, DIALOGUE, CONFIRMER, Demander, AJOUTER ENREGISTREMENT, MODIFIER ENREGISTREMENT, VISUALISER SELECTION et MODIFIER SELECTION.

Le tableau suivant résume, pour chaque type d'événement, l'appel des méthodes formulaire et objet :

Evénement	Méthode(s) objet	Méthode formulaire	Quel(s) objet(s)
Sur chargement	Oui	Oui	Tous
Sur libération	Oui	Oui	Tous
Sur validation	Oui	Oui	Tous
Sur clic	Oui (si cliquable) (*)	Oui	Seul l'objet impliqué
Sur double clic	Oui (si cliquable) (*)	Oui	Seul l'objet impliqué
Sur avant frappe clavier	Oui (si saisissable) (*)	Oui	Seul l'objet impliqué
Sur après frappe clavier	Oui (si saisissable) (*)	Oui	Seul l'objet impliqué
Sur après modification	Oui (si saisissable) (*)	Oui	Seul l'objet impliqué
Sur gain focus	Oui (si tabulable) (*)	Oui	Seul l'objet impliqué
Sur perte focus	Oui (si tabulable) (*)	Oui	Seul l'objet impliqué
Sur activation	Jamais	Oui	Aucun
Sur désactivation	Jamais	Oui	Aucun
Sur appel extérieur	Jamais	Oui	Aucun
Sur début glisser	Oui (si glissable) (**)	Oui	Seul l'objet impliqué
Sur déposer	Oui (si déposable) (**)	Oui	Seul l'objet impliqué
Sur glisser	Oui (si déposable) (**)	Jamais	Seul l'objet impliqué
Sur début survol	Oui	Oui	Tous
Sur survol	Oui	Oui	Tous
Sur fin survol	Oui	Oui	Tous
Sur menu sélectionné	Jamais	Oui	Aucun
Sur données modifiées	Oui (si modifiable) (*)	Oui	Seul l'objet impliqué
Sur appel zone du plug in	Oui	Oui	Seul l'objet impliqué
Sur entête	Oui	Oui	Tous
Sur impression corps	Oui	Oui	Tous
Sur impression sous total	Oui	Oui	Tous
Sur impression pied de page	Oui	Oui	Tous
Sur case de fermeture	Jamais	Oui	Aucun
Sur affichage corps	Oui	Oui	Tous
Sur ouverture corps	Jamais	Oui	Aucun
Sur fermeture corps	Jamais	Oui	Aucun
Sur redimensionnement	Jamais	Oui	Aucun
Sur nouvelle sélection	Oui (***)	Oui	Seul l'objet impliqué
Sur chargement ligne	Jamais	Oui	Aucun
Sur minuteur	Jamais	Oui	Aucun
Sur avant saisie	Oui (List box)	Jamais	Seul l'objet impliqué
Sur déplacement colonne	Oui (List box)	Jamais	Seul l'objet impliqué
Sur déplacement ligne	Oui (List box)	Jamais	Seul l'objet impliqué
Sur redimensionnement colonne	Oui (List box)	Jamais	Seul l'objet impliqué

Sur clic entête	Oui (List box)	Jamais	Seul l'objet impliqué
Sur après tri	Oui (List box)	Jamais	Seul l'objet impliqué
Sur clic long	Oui (Bouton 3D)	Oui	Seul l'objet impliqué
Sur clic flèche	Oui (Bouton 3D)	Oui	Seul l'objet impliqué
Sur déployer	Oui (Liste hiér.)	Jamais	Seul l'objet impliqué
Sur contracter	Oui (Liste hiér.)	Jamais	Seul l'objet impliqué
Sur chargement ressource URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur début chargement URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur erreur chargement URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur filtrage URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur fin chargement URL	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur ouverture lien externe	Oui (Zone Web)	Jamais	Seul l'objet impliqué
Sur refus ouverture fenêtre	Oui (Zone Web)	Jamais	Seul l'objet impliqué

(*) Référez-vous ci-dessous au paragraphe "Evénements, objets et propriétés" pour plus d'informations.
(**) Référez-vous à la section "Présentation du Glisser-Déposer" pour plus d'informations.
(***) Seuls les objets de type List box, Liste hiérarchique et Sous-formulaire prennent en charge cet événement.

IMPORTANT : Gardez constamment à l'esprit que, pour chaque événement, la méthode d'un formulaire ou d'un objet est appelée si l'événement correspondant a été sélectionné pour le formulaire ou l'objet. L'avantage de désactiver des événements en mode Développement (en utilisant la Liste des propriétés de l'éditeur de formulaires) est que vous pouvez réduire de manière très importante le nombre d'appels aux méthodes et donc optimiser la vitesse d'exécution des formulaires.

Evénements, objets et propriétés

Une méthode objet est appelée si l'événement peut réellement se produire pour l'objet en fonction de sa nature et de ses propriétés. Ce paragraphe détaille les événements à utiliser pour gérer les différents types d'objets.
A noter que la Liste des propriétés de l'éditeur de formulaires n'affiche que les événements compatibles avec l'objet sélectionné ou le formulaire.

Objets cliquables

Les objets cliquables sont gérés principalement avec la souris. Ces objets sont les suivants :

- Variables ou champs saisissables de type Booléen
- Boutons, boutons par défaut, boutons radio, cases à cocher, grilles de boutons
- Boutons 3D, boutons radio 3D, cases à cocher 3D
- Pop up menus, pop up menus hiérarchiques, menus Images

• Listes déroulantes, menus,
• Zones de défilement, listes hiérarchiques, list box
• Boutons invisibles, boutons inversés, boutons radio image
• Thermomètres, règles, cadrans
• Onglets,
• Séparateurs.

Lorsque l'événement Sur clic ou Sur double clic est sélectionné pour un de ces objets, vous pouvez détecter et gérer les clics sur l'objet à l'aide de la la commande Evenement formulaire qui retourne Sur clic ou Sur double clic selon le cas.
Si les deux événements sont sélectionnés pour un même objet, les événements Sur clic puis Sur double clic seront générés en cas de double-clic sur l'objet.

Pour tous les objets cliquables, l'événement Sur clic se produit une fois que le bouton de la souris est relâché. Il y a cependant des exceptions :
• Avec les boutons invisibles et les onglets, l'événement Sur clic se produit dès qu'un clic a été détecté — sans attendre le relâchement du bouton de la souris.
• Avec les thermomètres, règles et cadrans, si le format d'affichage indique que la méthode objet doit être appelée pendant que vous faites glisser les curseurs de contrôle, l'événement Sur clic survient dès que le clic est détecté.
• Avec les boutons 3D, l'événement Sur clic flèche est généré lorsque l'utilisateur clique sur la "flèche" (dès que le bouton de la souris est enfoncé), voir ci-dessous.

Note : Quelques objets peuvent être activés par le clavier. Une case à cocher, par exemple, une fois qu'elle a le focus, peut être sélectionnée à l'aide de la barre d'espace. Dans ce cas, l'événement Sur clic est quand même généré.

ATTENTION : Les combo-boxes ne sont pas considérées comme des objets cliquables. Une combo-box doit être perçue comme une zone de texte saisissable dont la liste déroulante fournit les valeurs par défaut. Par conséquent, vous gérez la saisie des données dans une combo-box à l'aide des événements Sur avant frappe clavier, Sur après frappe clavier et Sur données modifiées.

Objets saisissables par clavier

Les objets saisissables par clavier sont des objets dans lesquels vous saisissez des données par le clavier et pour lesquels vous pouvez filtrer les données au plus bas niveau en détectant les événements Sur après modification, Sur avant frappe clavier et Sur après frappe clavier.
Les objets et types de données saisissables sont les suivants :

• Tous les champs saisissables de type alpha, texte, date, heure, numérique ou (Sur après modification uniquement) image
• Toutes les variables saisissables de type alpha, texte, date, heure, numérique ou (Sur après modification uniquement) image

• Combo-boxes
• List boxes

Note : Bien qu'objets “saisissables”, les listes hiérarchiques ne gèrent pas les événements formulaire Sur après modification, Sur avant frappe clavier et Sur après frappe clavier (voir aussi le paragraphe “Listes hiérarchiques” ci-dessous).

• Sur avant frappe clavier et Sur après frappe clavier
Note : A compter de la version 2004.2 de 4D, l'événement Sur après frappe clavier peut généralement être avantageusement remplacé par l'événement Sur après modification (cf. ci-dessous).
Une fois que les événements Sur avant frappe clavier et Sur après frappe clavier ont été sélectionnés pour un objet, vous pouvez détecter et gérer la saisie par le clavier dans l'objet à l'aide de la commande Evenement formulaire qui va retourner Sur avant frappe clavier puis Sur après frappe clavier (pour plus d'informations, reportez-vous à la description de la commande Lire texte edite). Ces événements sont également activés par les commandes du langage simulant une action utilisateur, telles que GENERER FRAPPE CLAVIER.
A noter que les modifications des utilisateurs non effectuées via le clavier (coller, glisser-déposer, etc.) ne sont pas prises en compte. Pour traiter ces événements, vous devez utiliser Sur après modification.
Note : Les événements Sur avant frappe clavier et Sur après frappe clavier ne sont pas générés lors de l'utilisation d'une *méthode d'entrée*. Une méthode d'entrée (ou IME, Input Method Editor) est un programme ou un composant système permettant la saisie de caractères complexes ou de symboles (par exemple japonais ou chinois) à l'aide d'un clavier occidental.

• Sur après modification
Lorsqu’il est utilisé, cet événement est généré après chaque modification du contenu d’un objet saisissable, quelle que soit l’action à l’origine de cette modification, c'est-à-dire :
- les actions d’édition standard entraînant une modification du contenu telles que coller, couper, effacer ou annuler ;
- le déposer d’une valeur (action similaire au coller) ;
- toute saisie au clavier effectuée par l’utilisateur ; dans ce cas, l’événement Sur après modification est généré après les événements Sur avant frappe clavier et Sur après frappe clavier s’ils sont utilisés.
- une modification effectuée via une commande du langage simulant une action utilisateur (i.e. GENERER FRAPPE CLAVIER).
Attention, les actions suivantes ne déclenchent PAS cet événement :
- les actions d’édition ne modifiant pas le contenu de la zone, comme copier ou tout sélectionner ;
- le glisser d’une valeur (action similaire au copier) ;
- les modifications de contenu effectuées par programmation, à l’exception des commandes simulant une action utilisateur.

Cet événement permet de contrôler les actions utilisateur afin, par exemple, d'interdire de coller un texte trop volumineux, de filtrer certains caractères ou d'empêcher le couper d'un champ de mot de passe.

Objets modifiables

Les objets modifiables sont des objets ayant une source de données, dont la valeur peut être modifiée à l'aide de la souris ou du clavier, mais qui ne sont pas gérés par l'événement Sur clic. Ces objets sont les suivants :

- Tous les champs saisissables (sauf sous-tables et BLOB)
- Toutes les variables saisissables (sauf BLOB, pointeurs et tableaux)
- Combo-boxes
- Objets externes (pour lesquels la saisie de données est validée par le plug-in)
- Listes hiérarchiques
- List box

Ces objets reçoivent les événements Sur données modifiées. Lorsque la propriété d'événement Sur données modifiées est sélectionnée pour un de ces objets, vous pouvez détecter et gérer la modification de la valeur de la source de données à l'aide de la commande Evenement formulaire qui retourne Sur données modifiées. L'événement est généré dès que la variable associée à l'objet est mise à jour en interne par 4D (c'est-à-dire, en général, lorsque la zone de saisie de l'objet perd le focus).

Objets tabulables

Les objets tabulables sont ceux qui peuvent recevoir le focus lorsque vous utilisez la touche **Tab** et/ou si vous cliquez dessus. L'objet qui a le focus est celui qui reçoit les caractères saisis via le clavier et qui ne sont pas les *modificateurs* d'une commande de menu ou d'un objet tel qu'un bouton.

TOUS les objets sont tabulables SAUF ceux listés ci-dessous :

- Variables ou champs non saisissables
- Grilles de boutons
- Boutons 3D, boutons radio 3D, cases à cocher 3D
- Pop-up/listes déroulantes
- Menus déroulants hiérarchiques
- Pop-up menus Image
- Zones de défilement
- Boutons invisibles, boutons inversés, boutons radio Images
- Graphes
- Objets externes (pour lesquels la saisie de données est validée par le plug-in 4D)
- Onglets
- Séparateurs

Lorsque les événements Sur gain focus et/ou Sur perte focus sont sélectionnés pour un objet tabulable, vous pouvez détecter et gérer la modification du focus à l'aide de la commande Evenement formulaire qui retournera Sur gain focus ou Sur perte focus en fonction de l'événement.

Boutons 3D

Les boutons 3D autorisent la mise en place d'interfaces graphiques avancées (pour une description des boutons 3D, reportez-vous au manuel *Mode Développement*). Outre les événements génériques, deux événements spécifiques permettent de gérer de ces boutons :

• Sur clic long : cet événement est généré lorsqu'un bouton 3D reçoit un clic et que le bouton de la souris reste enfoncé pendant un certain laps de temps. En pratique, le délai à l'issue duquel l'événement est généré est égal au délai maximal séparant un double-clic, tel qu'il a été défini dans les préférences du système.
Cet événement peut être généré pour tous les styles de boutons 3D, boutons radio 3D et cases à cocher 3D, à l'exception des boutons 3D "ancienne génération" (style Décalage du fond) et des zones de flèche des boutons 3D avec pop up menu (cf. ci-dessous).
Cet événement est généralement utilisé pour afficher des pop up menus en cas de clics longs sur des boutons. L'événement Sur clic, s'il est coché, est généré si l'utilisateur relâche le bouton de la souris avant le délai du "clic long".

• Sur clic flèche : certains styles de boutons 3D peuvent être associés à un pop up menu et afficher une flèche. Un clic sur cette flèche fait généralement apparaître un menu de sélection proposant des actions supplémentaires en rapport avec l'action principale du bouton.
4D vous permet de gérer ce type de bouton à l'aide de l'événement Sur clic flèche. Cet événement est généré lorsque l'utilisateur clique sur la "flèche" (dès que le bouton de la souris est enfoncé) :
- si le pop up menu est de type "Séparé", l'événement est généré uniquement en cas de clic sur la zone fléchée du bouton.
- si le pop up menu est de type "Lié", l'événement est généré en cas de clic sur n'importe quelle partie du bouton. A noter qu'avec ce type de bouton, l'événement Sur clic long ne peut pas être généré.

Les styles de boutons 3D, boutons radio 3D et cases à cocher 3D acceptant la propriété "Avec pop up menu" sont : Aucun, Bouton barre outils, Bevel, Bevel arrondi et Office XP.

List box

Sept événements formulaires permettent de prendre en charge les spécificités des list box :

• Sur avant saisie : cet événement est généré juste avant qu'une cellule de list box passe en mode édition (c'est-à-dire, avant que le curseur de saisie soit affiché). Cet événement permet au développeur, par exemple, d'afficher un texte différent selon que l'utilisateur est en mode affichage ou édition.

• Sur nouvelle sélection : cet événement est généré à chaque fois que la sélection courante de lignes ou de colonnes de la list box est modifiée. Cet événement est également généré pour les listes d'enregistrements et les listes hiérarchiques.

• Sur déplacement colonne : cet événement est généré lorsqu'une colonne de list box est déplacée par l'utilisateur via le glisser-déposer. Il n'est pas généré si la colonne est glissée et déposée à son emplacement initial. La commande Numero colonne listbox deplacee permet de connaître le nouvel emplacement de la colonne.

• Sur déplacement ligne : cet événement est généré lorsqu'une ligne de list box est déplacée par l'utilisateur via le glisser-déposer. Il n'est pas généré si la ligne est glissée et déposée à son emplacement initial.

• Sur redimensionnement colonne : cet événement est généré lorsque la largeur d'une colonne de list box est modifiée (via la souris ou par programmation à l'aide de la commande FIXER LARGEUR COLONNE LISTBOX).

• Sur clic entête : cet événement est généré lorsqu'un clic se produit sur l'en-tête d'une colonne de list box. Dans ce cas, la commande Self permet de connaître l'en-tête de colonne sur laquelle le clic s'est produit. L'événement Sur clic est généré lorsqu'un clic droit (Windows) ou un Ctrl+clic (Mac OS) se produit sur une colonne ou un en-tête de colonne.
Si la propriété **Triable** a été cochée pour la list box, il est possible d'autoriser ou non le tri standard sur la colonne en passant la valeur 0 ou -1 dans la variable $0 :
- Si $0 vaut 0, le tri standard est effectué.
- Si $0 vaut -1, le tri standard n'est pas effectué et l'en-tête n'affiche pas la flèche de tri. Le développeur peut toutefois générer un tri des colonnes sur des critères personnalisés à l'aide des commandes de gestion des tableaux de 4D.
Si la propriété **Triable** n'a pas été cochée pour la list box, la variable $0 n'est pas utilisée.

• Sur après tri : cet événement est généré juste après qu'un tri standard ait été effectué (par conséquent, il n'est pas généré si $0 retourne -1 dans l'événement Sur clic entête). Ce mécanisme est utile pour conserver le sens du dernier tri effectué par l'utilisateur. Dans cet événement, la commande Self retourne un pointeur sur la variable de la colonne ayant été triée.

Listes hiérarchiques
Outre les événements génériques, trois événements formulaires spécifiques permettent de prendre en charge les actions utilisateurs effectuées sur les listes hiérarchiques :

• Sur nouvelle sélection : cet événement est généré à chaque fois que la sélection dans la liste hiérarchique est modifiée à la suite d'un clic ou de la frappe d'une touche au clavier. Cet événement est également généré dans les objets list box et les listes d'enregistrements.
• Sur déployer : cet événement est généré à chaque fois qu'un élément de la liste hiérarchique a été déployé via un clic ou une touche du clavier.
• Sur contracter : cet événement est généré à chaque fois qu'un élément de la liste hiérarchique a été contracté via un clic ou une touche du clavier.

Ces événements ne sont pas mutuellement exclusifs. Ils peuvent être générés les uns après les autres pour une liste hiérarchique :
- Suite à la frappe d'une touche clavier (dans l'ordre) :

Evénement	**Contexte**
Sur données modifiées	Un élément était en édition
Sur déployer / Sur contracter	Ouverture/fermeture de sous-liste à l'aide des touches fléchées -> ou <-
Sur nouvelle sélection	Sélection d'un nouvel élément
Sur clic	Activation de la liste par le clavier

- Suite à un clic souris (dans l'ordre) :

Evénement	**Contexte**
Sur données modifiées	Un élément était en édition
Sur déployer / Sur contracter	Ouverture/fermeture de sous-liste via un clic sur l'icône de déploiement/contraction *ou bien* Double-clic sur une sous-liste non éditable
Sur nouvelle sélection	Sélection d'un nouvel élément
Sur clic / Sur double clic	Activation de la liste par un clic ou un double-clic

Zones Web
Sept événements formulaires spécifiques sont disponibles pour les zones Web :

• Sur début chargement URL : cet événement est généré au début du chargement d'un nouvel URL dans la zone Web. La variable "*URL*" associée à la zone Web vous permet de connaître l'URL en cours de chargement.

Note : L'URL en cours de chargement est différent de l'URL courant (reportez-vous à la description de la commande WA Lire URL courant).

• Sur chargement ressource URL : cet événement est généré à chaque chargement d'une nouvelle ressource (image, frame, etc.) dans la page Web courante. La variable "*Progression du chargement*" associée à la zone vous permet de connaître l'état courant du chargement.

• Sur fin chargement URL : cet événement est généré lorsque toutes les ressources de l'URL courant ont été chargées. Vous pouvez appeler la commande WA Lire URL courant afin de connaître l'URL chargé.

• Sur erreur chargement URL : cet événement est généré lorsqu'une erreur a été détectée au cours du chargement d'un URL. Vous pouvez appeler la commande WA LIRE DERNIERE ERREUR URL afin d'obtenir des informations sur l'erreur.

• Sur filtrage URL : cet événement est généré lorsque le chargement d'un URL a été bloqué par la zone Web du fait d'un filtre mis en place via la commande WA FIXER FILTRES URL. Vous pouvez alors connaître l'URL bloqué à l'aide de la commande WA Lire dernier URL filtre.

• Sur ouverture lien externe : cet événement est généré lorsque le chargement d'un URL a été bloqué par la zone Web et que l'URL a été ouvert avec le navigateur courant du système, du fait d'un filtre mis en place via la commande WA FIXER FILTRES LIENS EXTERNES. Vous pouvez alors connaître l'URL bloqué à l'aide de la commande WA Lire dernier URL filtre.

• Sur refus ouverture fenêtre : cet événement est généré lorsque l'ouverture d'une fenêtre pop up a été bloquée par la zone Web. En effet, les zones Web 4D ne permettent pas l'ouverture de fenêtres pop up. Vous pouvez alors connaître l'URL bloqué à l'aide de la commande WA Lire dernier URL filtre.

Exemples

Dans tous les exemples ci-dessous, nous supposons que les propriétés d'événements des formulaires et des objets ont été sélectionnées de manière appropriée.

(1) L'exemple suivant trie une sélection de sous-enregistrements pour la sous-table [Parents]Enfants avant qu'un formulaire de la table [Parents] ne soit affiché à l'écran :

```
    ` Méthode d'un formulaire pour la table [Parents]
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TRIER SOUS ENREGISTREMENTS([Parents]Enfants;[Parents]Enfants'Prénom;>)
        ` ...
Fin de cas
```

(2) L'exemple suivant montre l'utilisation de l'événement Sur validation pour affecter automatiquement la date lorsque l'enregistrement est modifié :

```
    ` Méthode d'un formulaire
Au cas ou
        ` ...
    : (Evenement formulaire=Sur validation)
        [LaTable]Date de modification:=Date du jour
Fin de cas
```

(3) Dans l'exemple suivant, la gestion complète d'un menu déroulant (initialisation, clics et relâchement de l'objet) est placée dans la méthode de l'objet :

```
    ` Méthode objet du menu déroulant taTaille
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU ALPHA(31;taTaille;3)
        taTaille{1}:="Petit"
        taTaille{2}:="Moyen"
        taTaille{3}:="Grand"
    : (Evenement formulaire=Sur clic)
        Si (taTaille#0)
            ALERTE("Vous avez choisi la taille "+taTaille{taTaille}+".")
        Fin de si
    : (Evenement formulaire=Sur libération)
        EFFACER VARIABLE(taTaille)
Fin de cas
```

(4) L'exemple suivant montre comment, dans une méthode objet, gérer et valider l'opération de glisser-déposer à partir d'un champ qui n'accepte que des images.

```
    ` Méthode objet du champ Image [LaTable]uneImage
Au cas ou
    : (Evenement formulaire=Sur glisser)
            ` On est en train de glisser-déposer un objet et la souris est au-dessus d'un
            ` champ. Obtenir les informations sur l'objet source
        PROPRIETES GLISSER DEPOSER ($vpObjetSource;$vlElémentSource;
                                                        $lProcessSource)
            ` Notez que nous n'avons pas besoin de tester le numéro de process source
            ` pour la méthode objet exécutée parce qu'elle est dans le même process
        $vlTypeDonnées:=Type ($vpSrcObject->)
            ` Les données source sont-elles une image (champ, variable ou tableau) ?
        Si (($vlTypeDonnées=Est une image) | ($vlTypeDonnées=Est un tableau image))
                ` Accepter l'opération
            $0:=0
        Sinon
                ` Sinon, refuser l'opération
            $0:=-1
        Fin de si
```

```
: (Evenement formulaire=Sur déposer)
      ` Les données source ont été déposées sur l'objet, donc nous avons besoin de
      ` les copier dans l'objet. Obtenir les informations sur l'objet source
   PROPRIETES GLISSER DEPOSER ($vpObjetSource;$vlElémentSource;
                                                         $lProcessSource)
   $vlTypeDonnées:=Type ($vpSrcObject->)
   Au cas ou
         ` L'objet source est un champ ou une variable de type Image
      : ($vlTypeDonnées=Est une image)
            ` Est-ce que l'objet source est dans le même process (dans la même
            ` fenêtre et le même formulaire) ?
         Si ($lProcessSource=Numero du process courant)
               ` Copier la valeur source
            [LaTable]uneImage:=$vpObjetSource->
         Sinon
               ` Sinon, est-ce que l'objet source est une variable ?
            Si (Est une variable ($vpObjetSource))
                  ` Obtenir la valeur du process source
               LIRE VARIABLE PROCESS ($lProcessSource;$vpObjetSource->;
                                                         $vgImageGlissée)
               [LaTable]uneImage:=$vgImageGlissée
            Sinon
               ` Sinon, utiliser APPELER PROCESS pour obtenir la valeur du champ
               ` du process source
            Fin de si
         Fin de si
         ` L'objet source est un tableau de type Image
      : ($vlTypeDonnées=Est un tableau image)
            ` Est-ce que l'objet source est dans le même process (dans la même
            ` fenêtre et le même formulaire) ?
         Si ($lProcessSource=Numero du process courant)
               ` Copier la valeur source
            [LaTable]uneImage:=$vpSrcObject->{$vlElémentSource}
         Sinon
               ` Sinon, obtenir la valeur du process source
            LIRE VARIABLE PROCESS ($lProcessSource;$vpObjetSource
                  ->{$vlElémentSource};$vgImageGlissée)
            [LaTable]uneImage:=$vgImageGlissée
         Fin de si
   Fin de cas
Fin de cas
```

Note : Pour d'autres exemples sur la gestion des événements Sur glisser et Sur déposer, référez-vous aux exemples de la commande PROPRIETES GLISSER DEPOSER.

(5) L'exemple suivant est une méthode formulaire générique. Elle fait apparaître chacun des événements qui peuvent survenir lorsqu'un formulaire est utilisé comme formulaire sortie :

```
    ` Méthode formulaire d'un formulaire sortie
$vpFormTable:=Table du formulaire courant
Au cas ou
        ` ...
    : (Evenement formulaire=Sur entête)
            ` La zone en-tête va être imprimée ou affichée
        Au cas ou
            : (Avant selection($vpFormTable->))
                ` Le code pour la première rupture d'en-tête doit être placé ici
            : (Niveau = 1)
                ` Le code pour la rupture d'en-tête niveau 1 doit être placé ici
            : (Niveau = 2)
                ` Le code pour la rupture d'en-tête niveau 2 doit être placé ici
                ` ...
        Fin de cas
    : (Evenement formulaire=Sur impression corps)
            ` Un enregistrement va être imprimé
            ` Le code pour chaque enregistrement doit être placé ici
    : (Evenement formulaire=Sur impression sous total)
            ` Une rupture va être imprimée
        Au cas ou
            : (Niveau = 0)
                ` Le code pour la rupture 0 doit être placé ici
            : (Niveau = 1)
                ` Le code pour la rupture 1 doit être placé ici
                ` ...
        Fin de cas
    : (Evenement formulaire=Sur impression pied de page)
        Si (Fin de selection($vpFormTable->))
            ` Le code pour le dernier pied de page doit être placé ici
        Sinon
            ` Le code pour le pied de page doit être placé ici
        Fin de si
Fin de cas
```

(6) L'exemple suivant montre une méthode formulaire générique qui gère les événements pouvant survenir dans un formulaire sortie quand il s'affiche à l'aide de VISUALISER SELECTION ou MODIFIER SELECTION. Dans un but informatif, elle affiche l'événement dans la barre de titre de la fenêtre.

```
   ` Une méthode formulaire exemple
Au cas ou
   : (Evenement formulaire=Sur chargement)
      $vaEvénement:="Le formulaire va être affiché"
   : (Evenement formulaire=Sur libération)
      $vaEvénement:="Le formulaire sortie vient de se fermer et va disparaître de l'écran"
   : (Evenement formulaire=Sur affichage corps)
      $vaEvénement:="Affichage de l'enregistrement n°"+Chaine(Numero dans
                                                      selection([LaTable]))
   : (Evenement formulaire=Sur menu sélectionné)
      $vaEvénement:="Une commande de menu a été sélectionnée"
   : (Evenement formulaire=Sur entête)
      $vaEvénement:="L'en-tête va être imprimé ou affiché"
   : (Evenement formulaire=Sur clic)
      $vaEvénement:="On a cliqué sur un enregistrement"
   : (Evenement formulaire=Sur double clic)
      $vaEvénement:="On a double-cliqué sur un enregistrement"
   : (Evenement formulaire=Sur ouverture corps)
      $vaEvénement:="On a double-cliqué sur l'enregistrement n°"+Chaine
                                          (Numero dans selection([LaTable]))
   : (Evenement formulaire=Sur fermeture corps)
      $vaEvénement:="Retour au formulaire sortie"
   : (Evenement formulaire=Sur activation)
      $vaEvénement:="La fenêtre du formulaire passe au premier plan"
   : (Evenement formulaire=Sur désactivation)
      $vaEvénement:="La fenêtre du formulaire n'est plus au premier plan"
   : (Evenement formulaire=Sur menu sélectionné)
      $vaEvénement:="Une ligne de menu a été sélectionnée"
   : (Evenement formulaire=Sur appel extérieur)
      $vaEvénement:="Un appel extérieur a été reçu"
   Sinon
      $vaEvénement:="Que se passe-t-il ? L'événement n°"+Chaine(Evenement
                                                      formulaire)
Fin de cas
CHANGER TITRE FENETRE ($vaEvénement)
```

(7) Pour des exemples de gestion des événements Sur avant frappe clavier et Sur après frappe clavier, référez-vous aux exemples des commandes Lire texte edite, Frappe clavier et FILTRER FRAPPE CLAVIER.

(8) L'exemple suivant montre comment traiter de la même manière les clics et double-clics dans une zone de défilement :

```
    ` Méthode objet pour la zone de défilement taChoix
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU ALPHA (...;taChoix;...)
            ` ...
        taChoix:=0
    : ((Evenement formulaire=Sur clic) | (Evenement formulaire=Sur double clic))
        Si (taChoix#0)
                ` On a cliqué sur un élément, faire quelque chose
                ` ...
        Fin de si
            ` ...
Fin de cas
```

(9) L'exemple suivant montre comment traiter les clics et double-clics de manière différente (notez l'utilisation de l'élément zéro pour conserver la valeur de l'élément sélectionné) :

```
    ` Méthode objet pour la zone de défilement taChoix
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU ALPHA (...;taChoix;...)
            ` ...
        taChoix:=0
        taChoix{0}:="0"
    : (Evenement formulaire=Sur clic)
        Si (taChoix#0)
            Si (taChoix#Num(taChoix))
                    ` On a cliqué sur un élément, faire quelque chose
                    ` ...
                ` Sauvegarder l'élément nouvellement sélectionné pour la prochaine fois
                taChoix{0}:=Chaine(taChoix)
            Fin de si
        Sinon
            taChoix:=Num(taChoix{0})
             Fin de si
```

```
    : (Evenement formulaire=Sur double clic souris)
        Si (taChoix#0)
                ` On a double-cliqué sur un élément, faire quelque chose
        Fin de si
            ` ...
Fin de cas
```

(10) L'exemple suivant montre comment maintenir une zone contenant du texte à partir d'une méthode formulaire à l'aide des événements Sur gain focus et Sur perte focus :

```
    ` Méthode formulaire [Contacts];"Entrée"
Au cas ou
    : (Evenement formulaire=Sur chargement)
        C_TEXTE(vtZoneEtat)
        vtZoneEtat:=""
    : (Evenement formulaire=Sur gain focus)
        RESOUDRE POINTEUR (Objet focus;$vsNomVar;$vlNumTable;$vlNumChamp)
        Si (($vlNumTable#0) & ($vlNumChamp#0))
            Au cas ou
                : ($vlNumChamp=1) ` Champ nom
                    vtZoneEtat:="Saisissez le nom du contact, il sera automatiquement mis
                                                                        en majuscules."
                        ` ...
                : ($vlNumChamp=10) ` Champ code postal
                    vtZoneEtat:="Saisissez un code postal, il sera automatiquement vérifié
                                                                        et validé."
                        ` ...
            Fin de cas
        Fin de si
    : (Evenement formulaire=Sur perte focus)
        vtZoneEtat:=""
            ` ...
Fin de cas
```

(11) L'exemple suivant montre comment traiter l'événement de fermeture de fenêtre avec un formulaire utilisé pour l'entrée des données :

```
    ` Méthode pour un formulaire entrée
$vpFormulaireTable:=Table du formulaire courant
Au cas ou
        ` ...
```

```
    : (Evenement formulaire=Sur case de fermeture)
        Si (Enregistrement modifie($vpFormulaireTable->))
            CONFIRMER ("Cet enregistrement a été modifié. Voulez-vous sauvegarder les
                                                                    modifications ?")
            Si (OK=1)
                VALIDER
            Sinon
                NE PAS VALIDER
            Fin de si
        Sinon
            NE PAS VALIDER
        Fin de si
            ` ...
Fin de cas
```

(12) L'exemple suivant montre comment mettre en majuscules un champ Texte ou Alphanumérique chaque fois que la valeur est modifiée :

```
    ` Méthode objet pour [Contacts]Prénom
Au cas ou
        ` ...
    : (Evenement formulaire=Sur données modifiées)
        [Contacts]Prénom:= Majusc(Sous chaine([Contacts]Prénom;1;1))
                                        +Minusc(Sous chaine([Contacts]Prénom;2))
            ` ...
Fin de cas
```

Référence

APPELER PROCESS, FILTRER FRAPPE CLAVIER, FIXER MINUTEUR, Frappe clavier, Lire texte edite, PROPRIETES GLISSER DEPOSER, Table du formulaire courant.

FIXER MINUTEUR (tickCount)

Paramètre	Type		Description
tickCount	Entier long	→	Nombre de ticks ou -1 = Déclenchement dès que possible

Description

La commande FIXER MINUTEUR permet d'activer l'événement formulaire Sur minuteur et de fixer, pour le process courant, le nombre de ticks (1 tick = 1/60ème de seconde) entre chaque événement formulaire Sur minuteur.

Note : Pour plus d'informations sur cet événement formulaire, reportez-vous à la description de la commande Evenement formulaire.

Si elle est appelée dans un contexte autre que l'affichage d'un formulaire, cette commande ne fait rien.

Si vous passez -1 dans le paramètre tickCount, la commande activera l'événement formulaire Sur minuteur "dès que possible", autrement dit dès que l'application 4D rendra la main au gestionnaire d'événements. Ce principe permet notamment de s'assurer qu'un formulaire soit entièrement affiché avant de démarrer un traitement (fluidité de l'application).

Note serveur Web : Le serveur Web 4D peut tirer parti de cette commande ainsi que de l'événement formulaire Sur minuteur pour réafficher des formulaires 4D en mode contextuel. Ce fonctionnement permet d'obtenir l'envoi de pages HTML mises à jour "en temps réel", tout en économisant la bande passante. En effet, dans ce cas la mise à jour du formulaire n'est pas automatique, il vous faut pour cela appeler la commande REDESSINER. Il est donc possible d'optimiser le système en n'appelant REDESSINER que lorsque les données ont été modifiées.
Seuls les navigateurs qui interprètent le JavaScript permettront le redessinement automatique. La période définie par FIXER MINUTEUR sera utilisée par le navigateur ; elle doit être comprise entre quelques secondes (5 étant une valeur pratique) et le timeout du process Web. Reportez-vous à l'exemple n°2.

Pour inactiver par programmation le déclenchement de l'événement formulaire Sur minuteur, appelez de nouveau la commande FIXER MINUTEUR en passant 0 dans le paramètre nbTicks.

Exemples

(1) Vous souhaitez que, lorsqu'un formulaire est affiché à l'écran, un bip soit émis toutes les trois secondes. Pour cela, écrivez dans la méthode du formulaire :

```
Si (Evenement formulaire=Sur chargement)
   FIXER MINUTEUR(60*3)
Fin de si
...
Si (Evenement formulaire=Sur minuteur)
   BEEP
Fin de si
```

(2) Vous souhaitez que votre serveur Web provoque la mise à jour d'un formulaire 4D sur les navigateurs toutes les cinq secondes. Vous pouvez écrire, dans la méthode du formulaire :

```
Si (Evenement formulaire=Sur chargement)
   FIXER MINUTEUR(60*5)
Fin de si
...
Si (Evenement formulaire=Sur minuteur)
   ... `Vous pouvez placer ici un test sur la modification des données et
      `n'exécuter la ligne suivante que si les données ont été modifiées
   REDESSINER ([MaTable])
Fin de si
```

Référence

Evenement formulaire, REDESSINER.

Lire texte edite → Texte

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Texte	←	Texte en cours de saisie

Description

La commande Lire texte edite retourne le texte en cours de saisie dans un objet de formulaire.
Cette commande est principalement destinée à être utilisée avec le nouvel événement formulaire Sur après frappe clavier pour récupérer le texte au fur et à mesure de la frappe. Elle peut également être utilisée avec l'événement formulaire Sur avant frappe clavier.

Note : Pour des raisons d'harmonisation avec le nouvel événement formulaire Sur après frappe clavier, l'événement Sur entrée clavier a été renommé en Sur avant frappe clavier à compter de la version 6.5 de 4D.

La combinaison de cette commande avec les événements formulaire Sur avant frappe clavier et Sur après frappe clavier fonctionne de la manière suivante :
• Dès qu'un caractère est tapé au clavier, l'événement Sur avant frappe clavier est généré. Dans cet événement, la fonction Lire texte edite retourne le contenu de la zone avant la dernière frappe clavier. Par exemple, si la zone contient "PA" et que l'utilisateur tape "R", Lire texte edite retourne "PA" dans l'événement Sur avant frappe clavier. Si la zone ne contient rien au départ, Lire texte edite retourne une chaîne vide.
• Ensuite, l'événement formulaire Sur après frappe clavier est généré. Dans cet événement, la fonction Lire texte edite retourne le contenu de la zone y compris le dernier caractère entré au clavier. Par exemple, si la zone contient "PA" et que l'utilisateur tape "R", Lire texte edite retourne "PAR" dans l'événement Sur après frappe clavier.

Ces deux événements ne sont générés que dans les méthodes des objets concernés.
Dans un contexte autre que la saisie dans un formulaire, cette fonction retourne une chaîne vide.

Exemples

(1) Dans un formulaire entrée, vous souhaitez que les caractères saisis soient automatiquement mis en majuscules :

```
Si (Evenement formulaire=Sur apres frappe clavier)
   [Voyages]Agences:=Majusc(Lire texte edite)
Fin de si
```

(2) Voici un exemple de traitement à la volée des caractères saisis dans un champ texte. Le principe consiste à placer dans un autre champ texte (appelé “Mots”) la décomposition en mots de la phrase en cours de saisie. Pour cela, écrivez dans la méthode objet du champ de saisie :

```
Si (Evenement formulaire=Sur apres frappe clavier)
   $SaisieTempsRéel:=Lire texte edite
   PROPRIETES PLATE FORME ($plate_forme)
   Si ($plate_forme#3)   ` Macintosh ou Power Macintosh
      Repeter
         $PhraseDécomposée:=Remplacer chaine($SaisieTempsRéel;Caractere(32);
                                                          Caractere(13))
      Jusque (Position(" ";$PhraseDécomposée)=0)
   Sinon   ` Windows
      Repeter
      $PhraseDécomposée:=Remplacer chaine($SaisieTempsRéel;Caractere(32);
                                          Caractere(13)+Caractere(10))
      Jusque (Position(" ";$PhraseDécomposée)=0)
   Fin de si
   [Exemple]Mots:=$PhraseDécomposée
Fin de si
```

Note : Cet exemple n’est pas exhaustif puisque l’on considère que les mots sont séparés par des espaces uniquement (Caractere (32)). La mise au point d’un système complet nécessiterait l’ajout d’autres filtres afin de repérer tous les mots (point-virgules, virgules, apostrophes, etc...).

Référence

Evenement formulaire.

version 5

Note de compatibilité
Cette fonction n'est conservée que pour des raisons de compatibilité avec les versions précédentes de 4D. A compter de la version 6, il est préférable d'utiliser la fonction Evenement formulaire et de tester si elle retourne un événement tel que Sur clic.

Pendant → Booléen

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Si vous souhaitez que la phase Pendant du cycle d'exécution soit générée, assurez-vous qu'au moins un événement tel que Sur clic a bien été sélectionné, en mode Développement, dans les propriétés du formulaire et/ou des objets concernés.

Référence

Evenement formulaire.

20

Fenêtres

Le rôle des fenêtres est d'afficher des informations pour l'utilisateur. Trois actions principales nécessitent l'affichage d'une fenêtre : la saisie de données, l'affichage de données et l'affichage de messages destinés à l'utilisateur.

Il y a toujours au moins une fenêtre ouverte à l'écran. Si nécessaire, des barres de défilement sont ajoutées, afin de permettre à l'utilisateur de faire défiler le contenu d'un formulaire lorsque celui-ci est plus grand que la fenêtre. En mode Développement, cette fenêtre sera soit la fenêtre présentant vos enregistrements sous forme de liste (formulaire sortie), soit la fenêtre proposant un enregistrement en saisie (formulaire entrée). En mode Application, cette fenêtre sera l'écran présentant par défaut le logo de 4D.

Lorsque vous sélectionnez une commande de menu en mode Application, la fenêtre d'accueil peut être effacée et remplacée par des données lorsque vous faites appel aux commandes qui affichent des formulaires. Une fois que l'exécution de ces commandes est terminée, l'écran d'accueil apparaît de nouveau par défaut.

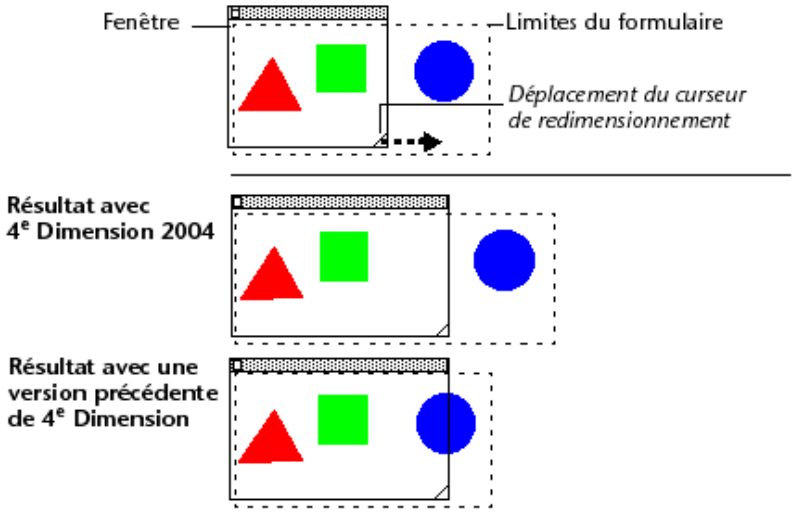
Pour créer vos propres fenêtres, faites appel à la commande Creer fenetre formulaire ou Creer fenetre. Il existe différents types de fenêtres personnalisées. Toutes les fenêtres ouvertes par ces commandes sont référencées au moyen de l'expression **RefFen**. Une RefFen identifie de façon unique une fenêtre ouverte. C'est une expression de type Entier long. Toutes les commandes fonctionnant avec des fenêtres personnalisées attendent une RefFen comme paramètre.

Lorsque vous n'en avez plus besoin, vous pouvez fermer une fenêtre personnalisée avec la commande FERMER FENETRE, ou en double-cliquant sur la case du menu Système (Windows) ou sur la case de fermeture (Mac OS), s'il y en a une.

Certaines commandes, telles que GRAPHE SUR SELECTION, QR ETAT ou encore IMPRIMER ETIQUETTES ouvrent leurs propres fenêtres et les placent au premier plan.

Si vous démarrez un nouveau process et n'ouvrez pas de fenêtre au démarrage de la méthode process, 4D en créera une automatiquement avec le type par défaut, dès qu'il sera nécessaire d'afficher un formulaire.

Bords pousseurs
A compter de la version 2004 de 4D, les bords droits et bas des fenêtres sont par défaut des séparateurs “pousseurs”. Cela signifie que les objets se trouvant à droite ou au-dessous des limites d’une fenêtre affichée à l’écran seront automatiquement repoussés vers la droite ou vers le bas en cas d’agrandissement de la fenêtre :



Ce mécanisme permet notamment de gérer des fenêtres à volets escamotables de type Explorateur (voir l'exemple de la commande FIXER TAILLE FORMULAIRE).

Note : Ce principe n’est pas mis en oeuvre lorsque la fenêtre comporte des barres de défilement.

Coordonnées des fenêtres et mode "droite à à gauche"
Dans les commandes de gestion des fenêtres, les coordonnées des fenêtres sont déterminées par rapport au point d'origine généralement situé en haut à gauche de la fenêtre/de l'écran. Toutefois, lorsque le mode "droite à gauche" est activé pour l'application, les coordonnées sont inversées et le point d’origine passe en haut à droite de la fenêtre/de l’écran. Par conséquent, dans ce mode les coordonnées horizontales manipulées par les commandes suivantes doivent être inversées :

Creer fenetre
Creer fenetre formulaire
Creer fenetre externe
COORDONNEES FENETRE
CHANGER COORDONNEES FENETRE
Chercher fenetre

Note : Pour plus d'informations sur le mode "droite à gauche", reportez-vous au manuel *Mode Développement* et à la description de la commande FIXER PARAMETRE BASE.

Référence

Creer fenetre, Creer fenetre formulaire, Types de fenêtres.

Vous spécifiez le type de fenêtre à ouvrir avec Creer fenetre à l'aide d'une des constantes prédéfinies suivantes :

Constante	Type	Valeur	Fenêtre flottante
Fenêtre standard	Entier long	8	Non
Fenêtre standard sans zoom	Entier long	0	Non
Fenêtre standard de taille fixe	Entier long	4	Non
Dialogue modal	Entier long	1	Non
Dialogue modal déplaçable	Entier long	5	Oui
Dialogue simple	Entier long	2	Oui
Dialogue ombré	Entier long	3	Oui
Fenêtre palette	Entier long	1984	Oui
Fenêtre à coins arrondis	Entier long	16	Non
Fenêtre pop up	Entier long	32	Non
Fenêtre feuille	Entier long	33	Non
Fenêtre feuille redim	Entier long	34	Non
Aspect métal	Entier long	2048	Oui
Mode compositing	Entier long	4096	Oui

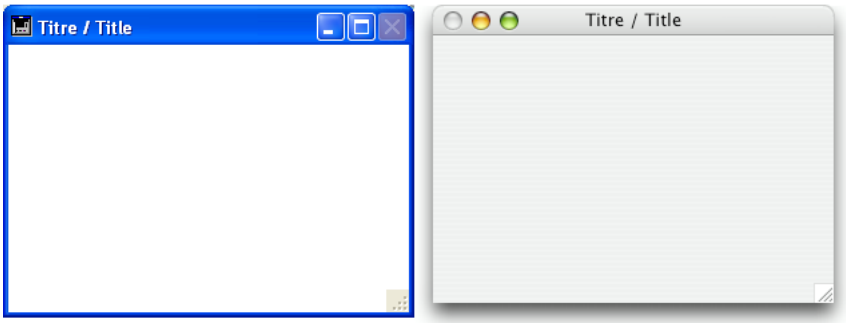
Fenêtres flottantes
Si vous passez une de ces constantes à Creer fenetre, vous créez une fenêtre standard. Pour ouvrir une fenêtre flottante, passez un type de fenêtre négatif à Creer fenetre.
Les fenêtres flottantes ont pour caractéristique principale de rester au premier plan même si l'utilisateur clique dans une autre fenêtre du process. Les fenêtres flottantes sont généralement utilisées pour afficher des informations permanentes ou des barres d'outils.

Fenêtres modales
Une fenêtre modale place l'utilisateur dans un état (ou “mode”) où il ne peut agir qu'à l'intérieur de la fenêtre. Tant qu'une fenêtre modale est affichée, les commandes de menu et les autres fenêtres de l'application sont inaccessibles. Pour refermer une fenêtre modale, l'utilisateur doit la valider, l'annuler ou utiliser une des options qu'elle propose. Les boîtes de dialogue d'alerte sont des exemples typiques de fenêtres modales.
Dans 4D, les fenêtres de type 1 et 5 sont modales.

Note : Une fenêtre modale reste toujours au premier plan. Par conséquent, lorsqu'une fenêtre modale appelle une fenêtre non modale, cette dernière s'affiche en arrière-plan, bien qu'elle ait été appelée postérieurement. Il ne faut donc pas effectuer ce type d'opération.
En revanche, lorsqu'une fenêtre modale appelle une autre fenêtre modale, cette dernière s'affiche au premier plan

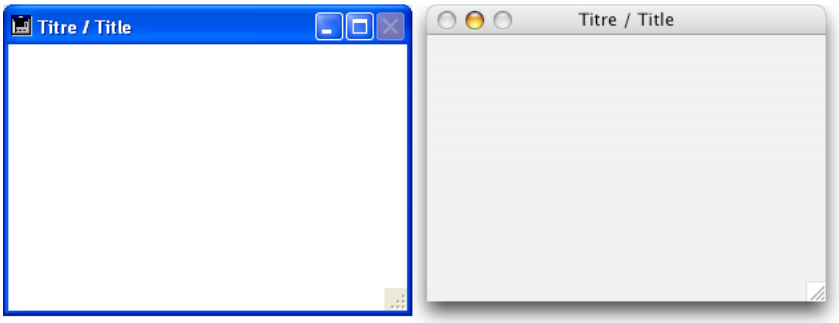
Vous trouverez ci-dessous une description de chaque type de fenêtre, sous Windows (à gauche) et Mac OS (à droite).

Fenêtre standard (8)



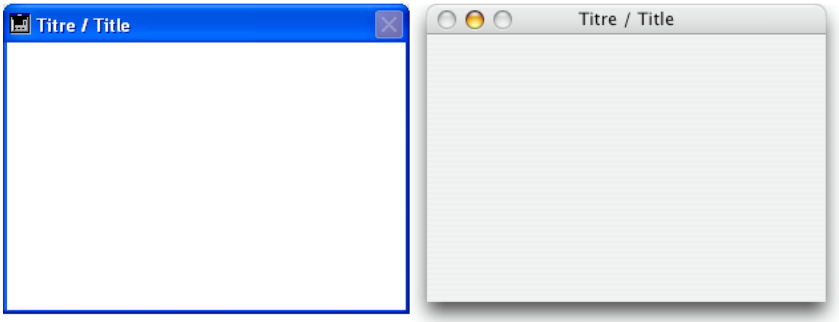
- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Oui
- Peut être redimensionnée : Oui
- Peut être agrandie/réduite ou "zoomée" : Oui
- Adaptée aux barres de défilement : Oui
- Utilisation : saisie de données avec des barres de défilement, VISUALISER SELECTION, MODIFIER SELECTION, etc.

Fenêtre standard sans zoom (0)



• Peut avoir un titre : Oui
• Peut comporter une case de fermeture ou un équivalent : Oui
• Peut être redimensionnée : Oui
• Peut être agrandie/réduite ou "zoomée" : Non sous Mac OS
• Adaptée aux barres de défilement : Oui
• Utilisation : saisie de données avec des barres de défilement, VISUALISER SELECTION, MODIFIER SELECTION, etc.

Fenêtre standard de taille fixe (4)

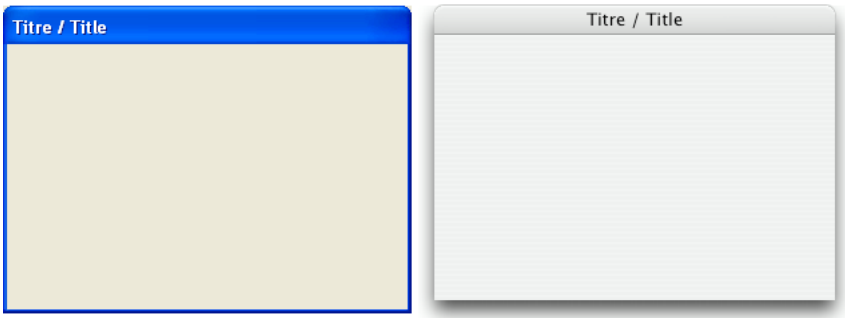


• Peut avoir un titre : Oui
• Peut comporter une case de fermeture ou un équivalent : Oui
• Peut être redimensionnée : Non sous Mac OS
• Peut être agrandie/réduite ou "zoomée" : Non
• Adaptée aux barres de défilement : Oui et Non
• Utilisation : saisie de données par AJOUTER ENREGISTREMENT(...;...*) ou équivalent

Dialogue modal (1)

- Peut avoir un titre : Non
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : DIALOGUE, AJOUTER ENREGISTREMENT(...;...*) ou équivalent
- Les fenêtres de ce type sont modales

Dialogue modal déplaçable (5)



- Peut avoir un titre : Oui
- Peut comporter une case de fermeture ou un équivalent : Non
- Peut être redimensionnée : Non
- Peut être agrandie/réduite ou "zoomée" : Non
- Adaptée aux barres de défilement : Non
- Utilisation : DIALOGUE, AJOUTER ENREGISTREMENT(...;...*) ou équivalent
- Les fenêtres de ce type sont modales mais peuvent être déplacées et utilisées comme fenêtres flottantes

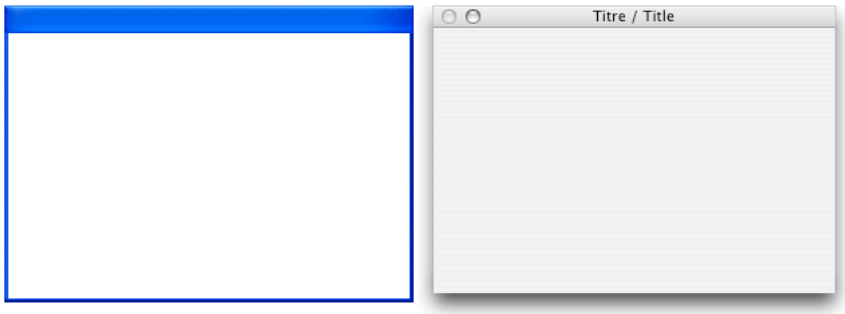
Dialogue ombré (3)

• Peut avoir un titre : Non
• Peut comporter une case de fermeture ou un équivalent : Non
• Peut être redimensionnée : Non
• Peut être agrandie/réduite ou "zoomée" : Non
• Adaptée aux barres de défilement : Non
• Utilisation : DIALOGUE, AJOUTER ENREGISTREMENT(...;...*) ou équivalent, sous Mac OS (non standard sous Windows).

Dialogue simple (2)

• Peut avoir un titre : Non
• Peut comporter une case de fermeture ou un équivalent : Non
• Peut être redimensionnée : Non
• Peut être agrandie/réduite ou "zoomée" : Non
• Adaptée aux barres de défilement : Non
• Utilisation : DIALOGUE, AJOUTER ENREGISTREMENT(...;...*) ou équivalent, sous Mac OS (non standard sous Windows).

Fenêtre palette (1984 {+ 1} {+ 2} {+ 4} {+ 8})

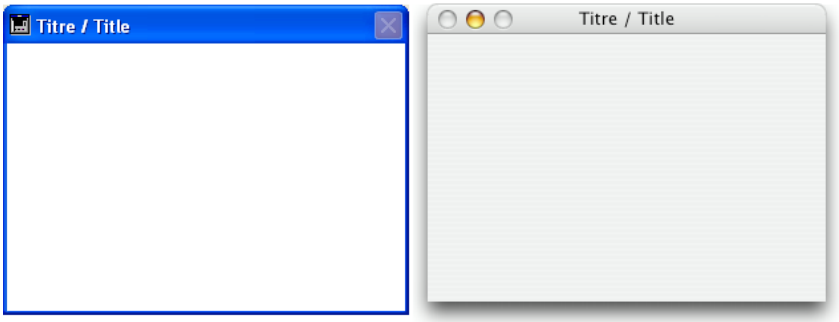


Lorsque vous appelez Creer fenetre, vous pouvez additionner une ou plusieurs constantes supplémentaires (listées ci-dessous) à la constante Fenêtre palette afin de créer des variantes ayant des comportements différents :

Constante	Type	Valeur
Avec case de zoom	Entier long	8
Avec case de contrôle de taille	Entier long	4
Avec titre de fenêtre	Entier long	2
Avec barre de titre active	Entier long	1

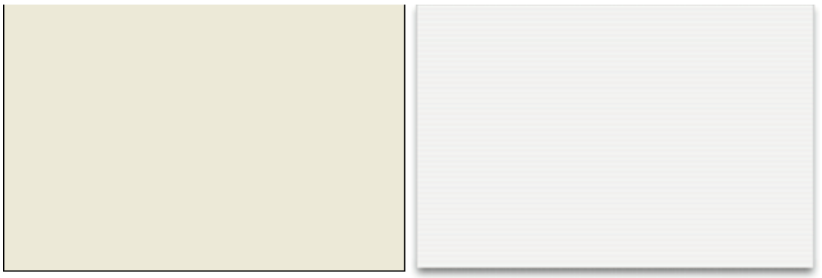
• Peut avoir un titre : Oui si la variante Avec titre de fenêtre est spécifiée
• Peut comporter une case de fermeture ou un équivalent : Oui
• Peut être redimensionnée : Oui si la variante Avec case de contrôle de taille est spécifiée
• Peut être agrandie/réduite ou "zoomée" : Oui si la variante Avec case de zoom est spécifiée
• Adaptée aux barres de défilement : Oui si la variante Avec case de contrôle de taille est spécifiée
• Utilisation : fenêtres flottantes avec DIALOGUE ou VISUALISER SELECTION (pas de saisie de données).

Fenêtre à coins arrondis (16)



• Peut avoir un titre : Oui
• Peut comporter une case de fermeture ou un équivalent : Oui
• Peut être redimensionnée : Non sous Mac OS
• Peut être agrandie/réduite ou "zoomée" : Non
• Adaptée aux barres de défilement : Non
• Utilisation : rare (obsolète)

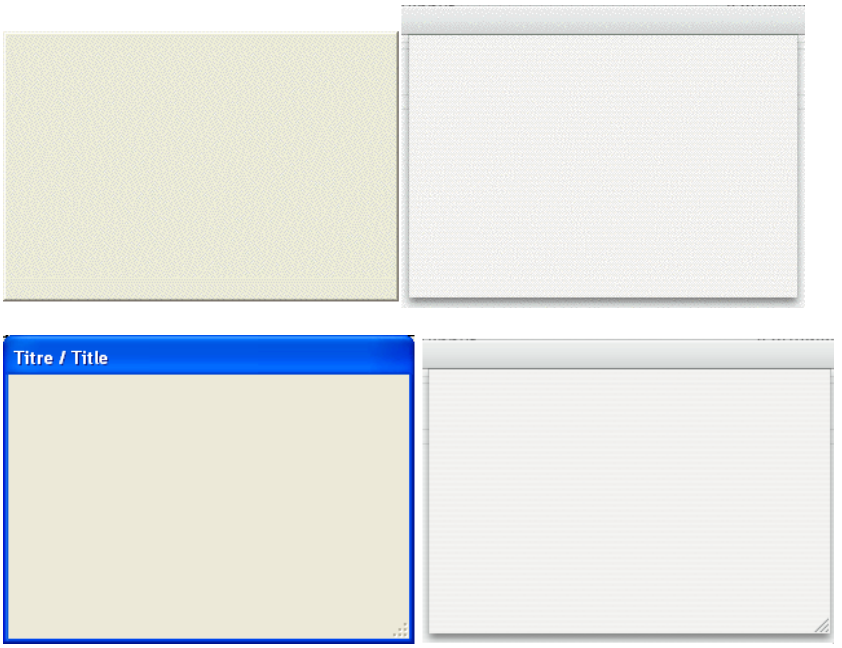
Fenêtre pop up (32)



Ce type de fenêtre reprend les caractéristiques essentielles du type Dialogue simple (2) et dispose de propriétés avancées spécifiques :
• La fenêtre est automatiquement refermée avec annulation lorsque :
- un clic se produit en-dehors de la fenêtre ;
- la fenêtre d'arrière-plan ou la fenêtre MDI est déplacée ;
- l'utilisateur appuie sur la touche **Echap** (ou **Esc**).
• Cette fenêtre s'affiche devant une fenêtre "parente" (elle ne doit d'ailleurs pas être utilisée comme fenêtre principale d'un process). La fenêtre d'arrière-plan n'est pas désactivée. En revanche, elle ne reçoit plus d'événement.

• Il n’est pas possible de redimensionner ou de déplacer la fenêtre à l’aide de la souris ; toutefois, lorsque cette opération est effectuée par programmation, le redessinement des éléments d’arrière-plan est optimisé.
• Utilisation : ce type de fenêtre est particulièrement adapté à la prise en charge des pop up menus associés aux boutons 3D de type “bevel” ou “barres outils”.

Fenêtre feuille (33) et **Fenêtre feuille redim** (34)

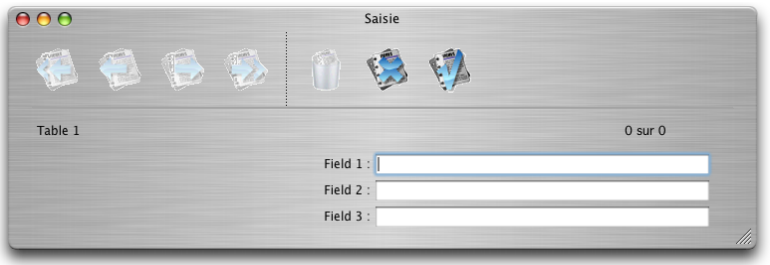


Les fenêtres feuilles (sheet windows) sont des fenêtres spécifiques de l’interface Mac OS X. Ces fenêtres “descendent” de la barre de titre de la fenêtre principale via une animation et s’affichent par-dessus celle-ci. Elles sont automatiquement centrées dans la fenêtre principale. Leurs propriétés sont comparables à celles des boîtes de dialogue modales. Elles sont généralement utilisées pour effectuer une action en relation directe avec celle se déroulant dans la fenêtre principale.

• Il n’est possible de créer une fenêtre feuille sous Mac OS X que si la dernière fenêtre ouverte est visible et de type document (formulaire).
• La commande crée une fenêtre de type 1 (Dialogue modal) au lieu du type 33 et de type 8 (Fenêtre standard) au lieu du type 34 :
- si la dernière fenêtre ouverte n’est pas visible ou n’est pas de type document,
- sous Windows.

• Comme une fenêtre feuille doit être dessinée par-dessus un formulaire, son affichage est repoussé dans l'événement Sur chargement du premier formulaire chargé dans la fenêtre (cf. exemple 4 de la commande Creer fenetre).
• Utilisation : DIALOGUE, AJOUTER ENREGISTREMENT(...;...*) ou équivalent, sous Mac OS (non standard sous Windows).

Aspect métal (2048)



Sous Mac OS, il est possible d'appliquer l'apparence "métal" aux fenêtres. Ce type d'apparence est largement répandu dans l'interface Macintosh. Sous Windows, cette propriété est sans effet.

Pour appliquer l'apparence "métal" à une fenêtre générée par la commande Creer fenetre, il suffit d'ajouter la constante Aspect métal au type de fenêtre défini dans le paramètre type. Par exemple :

```
$fen:=Creer fenetre(10;80;-1;-1;Fenêtre standard+Aspect métal;"")
```

Cette apparence peut être associée aux types de fenêtres suivants :

Fenêtre standard
Fenêtre standard sans zoom
Fenêtre standard de taille fixe
Dialogue modal déplaçable
Fenêtre à coins arrondis

Mode compositing (4096)

Le mode "compositing" est un mode de gestion interne des fenêtres sous Mac OS X, désormais préconisé par Apple. Dans 4D, ce mode est notamment requis pour l'affichage de certains objets dynamiques tels que les Zones Web.
Pour des raisons historiques, ce mode n'est pas utilisé dans toutes les fenêtres de 4D. Il est toujours utilisé par les formulaires projet exécutés depuis le mode Développement. En revanche, les fenêtres générées via les commandes Creer fenetre et Creer fenetre formulaire ne l'utilisent pas par défaut. Pour l'activer, il est nécessaire d'ajouter la constante Mode compositing (Creer fenetre) ou Form Mode compositing (Creer fenetre formulaire) au type de fenêtre lors de l'appel de la commande.
Sous Windows, cette propriété est sans effet.

Note : Certains objets d'ancienne architecture ne sont pas compatibles avec le mode compositing (par exemple les zones 4D Chart). S'ils sont affichés dans des fenêtres en mode compositing, ces objets ne fonctionneront pas.

Référence

Creer fenetre, Creer fenetre externe.

AFFICHER FENETRE

version 6.0.5

AFFICHER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande AFFICHER FENETRE permet d'afficher la fenêtre dont vous avez passé le numéro de référence dans fenêtre ou, si ce paramètre est omis, la fenêtre de premier plan du process courant.

La fenêtre doit au préalable avoir été cachée à l'aide de la commande CACHER FENETRE. Si la fenêtre est déjà affichée, la commande ne fait rien.

Exemple

Voir l'exemple de la commande CACHER FENETRE.

Référence

CACHER FENETRE.

CACHER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande CACHER FENETRE permet de masquer la fenêtre dont vous avez passé le numéro de référence dans fenêtre ou, si ce paramètre est omis, la fenêtre de premier plan du process courant. Cette commande vous permet, par exemple, dans un process comportant plusieurs fenêtres, de ne conserver à l'écran que la fenêtre active.

La fenêtre disparaît de l'écran mais reste ouverte. Vous pouvez continuer à lui appliquer par programmation tout traitement supporté par les fenêtres 4D.

Pour réafficher une fenêtre masquée par CACHER FENETRE :
• Utilisez la commande AFFICHER FENETRE et passez-lui le numéro de référence de la fenêtre.
• Ou bien, utilisez la page **Process** de l'Explorateur d'exécution. Sélectionnez le process dans lequel la fenêtre est gérée (process de gestion de la fenêtre ou Process principal) puis cliquez sur le bouton **Montrer**.

Si vous souhaitez cacher toutes les fenêtres d'un process, utilisez la commande CACHER PROCESS.

Exemple

Cet exemple est la méthode d'un bouton placé dans un formulaire entrée. Ce bouton ouvre une boîte de dialogue dans une nouvelle fenêtre du même process. Vous souhaitez masquer les autres fenêtres du process (un formulaire de saisie et une palette d'outils) afin de ne présenter que la boîte de dialogue. Une fois que celle-ci a été validée, vous réaffichez les fenêtres du process.

```
` Méthode objet du bouton "Informations"

CACHER FENETRE(Saisie)  ` Masquer la fenêtre de saisie
CACHER FENETRE(Palette)  ` Masquer la palette
$Infos:=Creer fenetre(20;100;500;400;8)  ` Créer la fenêtre d'informations
...  ` Placer ici les instructions nécessaires au remplissage et à la gestion du dialogue
```

```
FERMER FENETRE($Infos)  ` Fermer le dialogue
AFFICHER FENETRE(Saisie)
AFFICHER FENETRE(Palette)  ` Réafficher les autres fenêtres du process
```

Référence

AFFICHER FENETRE.

CHANGER COORDONNEES FENETRE

version 2004 (Modifiée)

CHANGER COORDONNEES FENETRE (gauche; haut; droite; bas{; fenêtre})

Paramètre	Type		Description
gauche	Numérique	→	Coordonnée gauche de l'intérieur de la fenêtre
haut	Numérique	→	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Numérique	→	Coordonnée droite de l'intérieur de la fenêtre
bas	Numérique	→	Coordonnée inférieure de l'intérieur de la fenêtre
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process si ce paramètre est omis

Description

La commande CHANGER COORDONNEES FENETRE modifie les coordonnées globales de la fenêtre dont vous avez passé le numéro de référence dans le paramètre fenêtre. Si la fenêtre n'existe pas, la commande ne fait rien.

Si vous omettez le paramètre fenêtre, CHANGER COORDONNEES FENETRE s'applique à la fenêtre de premier plan du process courant.

Ces coordonnées doivent être exprimées relativement au coin supérieur gauche de la zone de contenu de la fenêtre d'application (sous Windows) ou de l'écran principal (sous Mac OS). Les coordonnées décrivent le rectangle correspondant à la zone de contenu de la fenêtre (en excluant les barres de titres et les bordures).

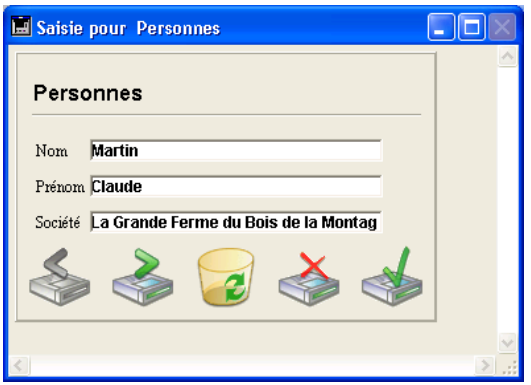
Attention : Utilisez cette commande avec précaution, car vous pouvez déplacer une fenêtre en-dehors des limites de la fenêtre principale (sous Windows) ou de l'écran (sous Mac OS). Pour éviter cela, vous pouvez utiliser des fonctions telles que Largeur ecran et Hauteur ecran pour bien vérifier les nouvelles coordonnées de la fenêtre.

Cette commande n'affecte pas les objets du formulaire. Si la fenêtre contient un formulaire, les objets du formulaire ne sont pas déplacés ou redimensionnés par la commande (quelles que soient leurs propriétés). Seule la fenêtre est modifiée. Pour modifier une fenêtre de formulaire en tenant compte de ses propriétés de redimensionnement et des objets qu'elle contient, vous devez utiliser la commande REDIMENSIONNER FENETRE FORMULAIRE.

Exemples

(1) Reportez-vous à l'exemple de la commande LISTE FENETRES.

(2) Soit la fenêtre suivante :



Après l'exécution de la ligne suivante :

CHANGER COORDONNEES FENETRE(100;100;300;300)

La fenêtre apparaît ainsi :

Référence

COORDONNEES FENETRE, DEPLACER FENETRE, REDIMENSIONNER FENETRE FORMULAIRE.

CHANGER TITRE FENETRE

version 6.0 (Modifiée)

CHANGER TITRE FENETRE (titre{; fenêtre})

Paramètre	Type		Description
titre	Alpha	→	Titre de la fenêtre
fenêtre	RefFen	→	Numéro de référence de fenêtre ou Fenêtre au premier plan du process courant si ce paramètre est omis

Description

La commande CHANGER TITRE FENETRE remplace le titre de la fenêtre dont le numéro de référence est passé dans fenêtre par le texte passé dans titre (longueur maximale 80 caractères).

Si la fenêtre n'existe pas, CHANGER TITRE FENETRE ne fait rien.

Si vous omettez le paramètre fenêtre, CHANGER TITRE FENETRE remplace le titre de la fenêtre de premier plan du process courant.

Note : En mode Développement, 4D définit automatiquement les titres des fenêtres — par exemple “Saisie pour table1” est affiché lorsque vous passez en saisie de données. Si vous changez le titre d'une fenêtre du mode Développement, il est probable que 4D le remplacera par la suite. En revanche, en mode Application, 4D ne modifie pas le titre des fenêtres.

Exemple

Vous effectuez une saisie dans un formulaire et vous cliquez sur un bouton qui déclenche une longue opération (par exemple une modification par programmation des enregistrements liés affichés dans un sous-formulaire). Vous pouvez afficher des informations sur la progression des opérations dans le titre de la fenêtre :

```
    ` Méthode objet du bouton bAnalyse
Au cas ou
    : (Evenement formulaire=Sur clic)
            ` Sauvegarde du titre courant de la fenêtre dans une variable
        $vsTitreCour:=Titre fenetre
            ` Commencer l'opération longue
        DEBUT SELECTION([Lignes facture])
        Boucle($vlRecord;1;Enregistrements trouves([Lignes facture]))
            FAIRE QUELQUE CHOSE
                ` Afficher la progression
            CHANGER TITRE FENETRE("Traitement de la ligne #"+Chaine($vlEnreg))
        Fin de si
            ` Remettre en place l'ancien titre de fenêtre
        CHANGER TITRE FENETRE($vsTitreCour)
Fin de cas
```

Référence

Titre fenetre.

Chercher fenetre

Chercher fenetre (gauche; haut{; partieFenêtre}) → RefFen

Paramètre	Type		Description
gauche	Numérique	→	Coordonnée globale gauche
haut	Numérique	→	Coordonnée globale supérieure
partieFenêtre	Numérique	←	Numéro de partie de fenêtre
Résultat	RefFen	←	Numéro de référence de fenêtre

Description

La commande Chercher fenetre retourne (s'il existe) le numéro de référence de la première fenêtre "touchée" par le point dont vous passez les coordonnées dans gauche et haut.

Ces coordonnées doivent être exprimées relativement au coin supérieur gauche de la zone de contenu (l'intérieur) de la fenêtre d'application (sous Windows) ou de l'écran principal (sous Mac OS).

Si vous passez le paramètre partieFenêtre, qu'une fenêtre ait été trouvée ou non, ce paramètre retournera une des constantes prédéfinies suivantes :

Constantes	Type	Valeur	Plate-forme
Dans barre de menu	Entier long	1	Macintosh
Dans fenêtre système	Entier long	2	Macintosh
Dans zone contenu	Entier long	3	Windows ou Macintosh
Dans barre de titre	Entier long	4	Macintosh
Dans case de contrôle de taille	Entier long	5	Macintosh
Dans case de fermeture	Entier long	6	Macintosh
Dans case de zoom	Entier long	8	Macintosh

Référence

Fenetre premier plan, Fenetre suivante.

COORDONNEES FENETRE

version 2003 (Modifiée)

COORDONNEES FENETRE (gauche; haut; droite; bas{; fenêtre})

Paramètre	Type		Description
gauche	Numérique	←	Coordonnée gauche de l'intérieur de la fenêtre
haut	Numérique	←	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Numérique	←	Coordonnée droite de l'intérieur de la fenêtre
bas	Numérique	←	Coordonnée inférieure de l'intérieur de la fenêtre
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process si omis ou Fenêtre MDI si -1 (Windows)

Description

La commande COORDONNEES FENETRE retourne les coordonnées globales de la fenêtre dont vous avez passé le numéro de référence dans le paramètre fenêtre. Si la fenêtre n'existe pas, les variables des paramètres sont inchangées.

Si vous omettez le paramètre fenêtre, COORDONNEES FENETRE s'applique à la fenêtre de premier plan du process courant.

Les coordonnées retournées sont exprimées relativement au coin supérieur gauche de la zone de contenu de la fenêtre d'application (sous Windows) ou de l'écran principal (sous Mac OS). Les coordonnées retournent le rectangle correspondant à la zone de contenu de la fenêtre (en excluant les barres de titres et les bordures).

Note : Sous Windows, si vous passez -1 dans fenêtre, COORDONNEES FENETRE retourne les coordonnées de la fenêtre d'application (fenêtre MDI) relativement à l'écran.

Exemple

Reportez-vous à l'exemple de la commande LISTE FENETRES.

Référence

CHANGER COORDONNEES FENETRE.

Creer fenetre (gauche; haut; droite; bas{; type{; titre{; caseFermeture}}}){ → RefFen }

Paramètre	Type		Description
gauche	Numérique	→	Coordonnée gauche de l'intérieur de la fenêtre
haut	Numérique	→	Coordonnée supérieure de l'intérieur de la fenêtre
droite	Numérique	→	Coordonnée droite de l'intérieur de la fenêtre
bas	Numérique	→	Coordonnée inférieure de l'intérieur de la fenêtre
type	Numérique	→	Type de la fenêtre
titre	Alpha	→	Titre de la fenêtre
caseFermeture	Alpha	→	Méthode à appeler en cas de double-clic sur la case du menu Système ou de clic sur la case de fermeture
Résultat	RefFen	←	Numéro de référence de la fenêtre

Description

Creer fenetre ouvre une nouvelle fenêtre dont les dimensions sont définies par les quatre premiers paramètres :
• gauche est la distance en pixels entre le côté gauche de la fenêtre de l'application et le côté intérieur gauche de la fenêtre.
• haut est la distance en pixels entre le haut de la fenêtre de l'application et le bord supérieur de l'intérieur de la fenêtre.
• droite est la distance en pixels entre le côté gauche de la fenêtre de l'application et le côté intérieur droit de la fenêtre.
• bas est la distance en pixels entre le haut de la fenêtre de l'application et le bord inférieur de l'intérieur de la fenêtre.

Si vous passez -1 dans droite et bas, vous indiquez à 4D qu'il faut redimensionner automatiquement la fenêtre si les conditions suivantes sont réunies :
• Vous avez conçu un formulaire et défini ses options de dimensionnement dans la fenêtre des propriétés des formulaires du mode Développement.
• Avant d'appeler Creer fenetre vous avez sélectionné le formulaire à l'aide de la commande FORMULAIRE ENTREE, à laquelle vous avez passé le paramètre optionnel *.

Important : Ce dimensionnement automatique de la fenêtre n'aura lieu que si vous avez au préalable appelé la commande FORMULAIRE ENTREE pour le formulaire que vous allez afficher dans la fenêtre, et si vous lui avez passé le paramètre optionnel *.

• Le paramètre type est optionnel. Il définit le type de fenêtre que vous souhaitez afficher, et correspond aux différentes fenêtres présentées dans la section Types de fenêtres. Si le type passé est négatif, la fenêtre sera flottante. Si le type n'est pas spécifié, le type 1 est utilisé par défaut.

• Le paramètre titre indique le titre (optionnel) de la fenêtre.

Si vous passez une chaîne de caractères vide ("") dans titre, vous indiquez à 4D d'utiliser les valeurs saisies dans la zone **Nom de la fenêtre** de la fenêtre des Propriétés du formulaire en mode Développement pour le titre du formulaire que vous allez afficher dans la fenêtre.

Important : Le nom par défaut du formulaire ne sera appliqué à la fenêtre que si vous avez appelé la commande FORMULAIRE ENTREE pour le formulaire que vous allez afficher dans la fenêtre et si vous lui avez passé le paramètre optionnel *.

• Le paramètre caseFermeture, optionnel, désigne la méthode de gestion de la fermeture de la fenêtre. Si ce paramètre est passé, la case du menu Système (sous Windows) ou une case de fermeture (sous Mac OS) est ajoutée à la fenêtre. Lorsque l'utilisateur Windows double-clique sur la case du menu Système ou que l'utilisateur Mac OS clique sur la case de fermeture, la méthode passée dans caseFermeture est exécutée.

Note : Vous pouvez aussi gérer la fermeture à partir de la méthode du formulaire affiché dans la fenêtre pendant l'événement Sur case de fermeture. Pour plus d'informations sur ce point, reportez-vous à la commande Evenement formulaire.

Si plusieurs fenêtres sont ouvertes dans le même process, la dernière fenêtre créée est la fenêtre active (de premier plan) du process. Seules les informations situées dans la fenêtre active peuvent être modifiées. Toutes les autres fenêtres peuvent être visualisées. Lorsque l'utilisateur tape une touche du clavier, la fenêtre active vient toujours se placer au premier plan, si elle n'y est pas déjà.

Les formulaires sont affichés à l'intérieur de fenêtres ouvertes à l'écran. Le texte passé à la commande MESSAGE est également affiché dans une fenêtre.

Exemples

(1) La méthode projet suivante ouvre une fenêtre centrée dans la fenêtre principale (sous Windows) ou dans l'écran principal (sous Mac OS). Notez qu'elle accepte deux, trois ou quatre paramètres :

```
` Méthode projet OUVRIR FENETRE CENTREE
` $1 – Largeur de la fenêtre
` $2 – Hauteur de la fenêtre
` $3 – Type de la fenêtre (optionnel)
` $4 – Titre de la fenêtre (optionnel)
$SW:=Largeur ecran\2
$SH:=(Hauteur ecran\2)-10
$WW:=$1\2
$WH:=$2\2
Au cas ou
    : (Nombre de parametres=2)
        Creer fenetre($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH)
    : (Nombre de parametres=3)
        Creer fenetre($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3)
    : (Nombre de parametres=4)
        Creer fenetre($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3;$4)
Fin de cas
```

Une fois que cette méthode projet est écrite, vous pouvez l'utiliser de la manière suivante :

```
OUVRIR FENETRE CENTREE (400;250;Dialogue simple;"Mise à jour Archives")
DIALOGUE([Table outils];"OPTIONS MAJ")
FERMER FENETRE
Si (OK=1)
    ` ...
Fin de si
```

(2) L'exemple suivant crée une fenêtre flottante comportant une case de menu système (sous Windows) ou une case de fermeture (sous Mac OS). La fenêtre est créée dans le coin supérieur droit de la fenêtre de l'application.

```
Creer fenetre(Largeur ecran-149;33;Largeur ecran-4;178;-Fenêtre palette;"";
                                                              "caseFermeture")
DIALOGUE([Dialogues];"Palette de couleurs")
```

La méthode caseFermeture appelle la commande NE PAS VALIDER :

```
NE PAS VALIDER
```

(3) L'exemple suivant ouvre une fenêtre dont le titre et la taille proviennent des propriétés du formulaire affiché dans la fenêtre :

```
FORMULAIRE ENTREE([Clients];"Ajout d'enregistrements";*)
Creer fenetre(10;80;-1;-1;Fenêtre standard;"")
Repeter
   AJOUTER ENREGISTREMENT([Clients])
Jusque (OK=0)
```

Rappel : Pour que la fonction Creer fenetre utilise automatiquement les propriétés du formulaire, vous devez avoir appelé FORMULAIRE ENTREE avec le paramètre optionnel * et les propriétés du formulaire doivent avoir été définies en fonction de cette utilisation.

(4) Cet exemple illustre le mécanisme de “retard” d’affichage des fenêtres feuille sous Mac OS X :

```
$maFenêtre:=Creer fenetre(10;10;400;400;Fenêtre feuille)
   `A cet instant la fenêtre est créée mais reste invisible
DIALOGUE([Table];"formDial")
   `L’événement Sur chargement est généré puis la fenêtre feuille est affichée, elle
   `"descend" du dessous de la barre de titre
```

Référence

Creer fenetre externe, Creer fenetre formulaire, FERMER FENETRE.

Creer fenetre externe (gauche; haut; droit; bas; type; titre; zonePlugin) → Numérique

Paramètre	Type		Description
gauche	Numérique	→	Coordonnée gauche de l'intérieur de la fenêtre
haut	Numérique	→	Coordonnée supérieure de l'intérieur de la fenêtre
droit	Numérique	→	Coordonnée droite de l'intérieur de la fenêtre
bas	Numérique	→	Coordonnée inférieure de l'intérieur de la fenêtre
type	Numérique	→	Type de la fenêtre
titre	Alpha	→	Titre de la fenêtre
zonePlugin	Alpha	→	Commande de zone externe
Résultat	Numérique	←	Numéro de référence de la fenêtre et de la zone externe

Description

Creer fenetre externe crée une nouvelle fenêtre et affiche la zone externe gérée par la commande zonePlugin fournie par un plug-in 4D. Le code à passer dans zonePlugin est généralement de la forme "_NomPlugin", par exemple : _4D Write, _4D View ou _4D Draw.

Creer fenetre externe retourne un Entier long qui peut être utilisé à la fois comme numéro de référence de fenêtre (qui peut être exploité par les autres commandes du thème Fenêtres) et comme référence de la zone externe affichée dans la fenêtre (qui peut être exploité par les autres routines du plug-in 4D).

Les six premiers paramètres sont identiques à ceux de la commande Creer fenetre. Cependant, aucun d'entre eux n'est optionnel.

Creer fenetre externe crée une fenêtre "sans mode", c'est-à-dire que la commande n'attend pas d'action de l'utilisateur, ce qui vous permet d'afficher plusieurs fenêtres actives simultanément. Vous pouvez naviguer parmi chaque fenêtre en cliquant dessus. Vous pouvez modifier celle qui se trouve au premier plan. Si le type de la fenêtre comporte une barre de titre, une case du menu Système (Windows) ou une case de fermeture (Macintosh) sera ajoutée à la fenêtre pour permettre à l'utilisateur de refermer la fenêtre.

Exemples

(1) L'exemple suivant ouvre une fenêtre externe et affiche une zone externe 4D Write :

```
wrWind:=Creer fenetre externe (50; 50; 350; 450; 8; "Ecrire lettre"; "_4D WRITE")
```

(2) L'exemple suivant referme la fenêtre externe ouverte dans l'exemple précédent :

```
FERMER FENETRE (wrWind)
```

Référence

Creer fenetre, FERMER FENETRE.

Creer fenetre formulaire

version 11 (Modifiée)

Creer fenetre formulaire ({laTable; }nomForm{; type{; posH{; posV{; *}}}}) → RefFen

Paramètre	Type		Description
laTable	Table	→	Table du formulaire ou Table par défaut si ce paramètre est omis
nomForm	Alpha	→	Nom du formulaire
type	Entier long	→	Type de la fenêtre
posH	Entier long	→	Position horizontale de la fenêtre
posV	Entier long	→	Position verticale de la fenêtre
*	*	→	Conserver la position et la taille précédentes de la fenêtre
Résultat	RefFen	←	Numéro de référence de la fenêtre

Description

La commande Creer fenetre formulaire crée une nouvelle fenêtre utilisant les propriétés de taille et de redimensionnement du formulaire nomForm, passé en paramètre.

A noter que le formulaire nomForm n'est pas affiché dans la fenêtre créée. Il vous appartient, si vous le souhaitez, d'afficher le formulaire (par exemple à l'aide de la commande AJOUTER ENREGISTREMENT).

Par défaut, si le paramètre type n'est pas passé, la fenêtre créée est de type standard et comporte une case de fermeture. A la différence de la commande Creer fenetre, aucune méthode n'est associée à cette case de fermeture : un clic sur la case de fermeture provoquera simplement l'annulation du formulaire, sauf si l'événement Sur case de fermeture est activé pour le formulaire, auquel cas le code associé à cet événement sera exécuté.

Si le formulaire nomForm est redimensionnable, la fenêtre créée comporte également une case de zoom et une case de contrôle de taille.

Note : Vous pouvez connaître les principales propriétés d'un formulaire à l'aide de la commande LIRE PROPRIETES FORMULAIRE.

• Le paramètre optionnel type vous permet de spécifier un type de fenêtre. Ce paramètre doit contenir une des constantes prédéfinies suivantes, placées dans le thème “Creer fenetre formulaire” :

Constante	Type	Valeur
Form fenêtre standard	Entier long	8
Form dialogue modal	Entier long	1
Form dialogue modal déplaçable	Entier long	5
Form fenêtre palette	Entier long	1984
Form fenêtre pop up	Entier long	32
Form fenêtre feuille	Entier long	33
Form Mode compositing	Entier long	4096

Notes :
• Les attributs de la fenêtre créée (case de contrôle de taille, case de fermeture...) dépendent des spécifications d'interface du système d'exploitation pour le type choisi. Il est donc possible d'obtenir des résultats différents en fonction de la plate-forme.
• La constante Form Mode compositing doit être ajoutée à l'une des autres constantes de type afin d'activer ce mode pour la fenêtre.
• Pour plus d'informations sur les types de fenêtres, reportez-vous à la section Types de fenêtres. A noter que seuls les types listés dans le thème “Creer fenetre formulaire” peuvent être utilisés avec la commande Creer fenetre formulaire.

• Le paramètre optionnel posH vous permet de définir l’emplacement horizontal de la fenêtre. Vous pouvez passer dans ce paramètre une coordonnée fixe exprimée en points (cf. commande Creer fenetre), ou l’une des constantes prédéfinies suivantes, placées dans le thème “Creer fenetre formulaire” :

Constante	Type	Valeur
Centrée horizontalement	Entier long	65536
A gauche	Entier long	131072
A droite	Entier long	196608

• Le paramètre optionnel posV vous permet de définir l’emplacement vertical de la fenêtre. Vous pouvez passer dans ce paramètre une coordonnée fixe exprimée en points (cf. commande Creer fenetre), ou l’une des constantes prédéfinies suivantes, placées dans le thème “Creer fenetre formulaire” :

Constante	Type	Valeur
Centrée verticalement	Entier long	262144
En haut	Entier long	327680
En bas	Entier long	393216

Ces paramètres tiennent compte de la présence de la barre d’outils, de la barre de menus, et de la taille courante de la fenêtre de l’application (sous Windows).

Si vous passez le paramètre optionnel *****, la position et la taille courantes de la fenêtre sont mémorisées au moment où elle est refermée. Lorsque la fenêtre est réouverte par la suite, elle conserve sa position et sa taille précédentes. Dans ce cas, les paramètres posV et posH ne sont utilisés que pour la première ouverture de la fenêtre.

Exemples

(1) L'instruction suivante ouvre une fenêtre standard avec case de fermeture automatiquement ajustée à la taille du formulaire "Entrée". Comme le formulaire a la propriété "Redimensionnable", la fenêtre comporte également une case de contrôle de taille et une case de zoom :

```
$refFen:=Creer fenetre formulaire ([Table1];"Entrée")
```

(2) L'instruction suivante ouvre, en haut et à gauche de l'écran, une palette flottante basée sur un formulaire projet nommé "Outils". Cette palette conservera sa précédente position à chaque nouvelle ouverture :

```
$refFen:=Creer fenetre formulaire ("Outils"; Form fenêtre palette; A gauche; En haut;*)
```

Référence

Creer fenetre, LIRE PROPRIETES FORMULAIRE, Types de fenêtres.

DEPLACER FENETRE

Paramètre	Type	Description

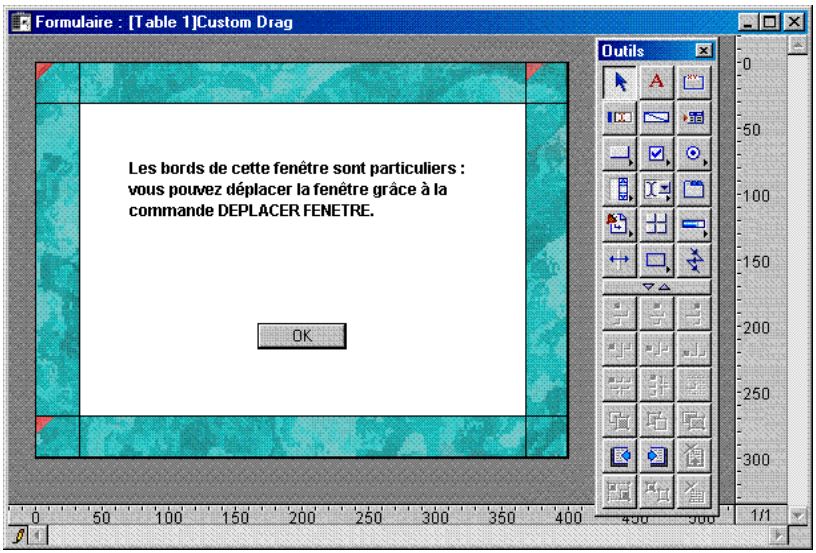
Cette commande ne requiert pas de paramètre

Description

La commande DEPLACER FENETRE permet de faire glisser la fenêtre dans laquelle l'utilisateur a cliqué puis de la déplacer en fonction des mouvements de la souris. Généralement, cette commande est appelée depuis la méthode d'un objet capable de répondre instantanément aux clics souris (par exemple un bouton invisible).

Exemple

Le formulaire suivant, présenté ici dans l'éditeur de formulaires, contient un fond créé par une image statique par-dessus laquelle quatre boutons invisibles ont été placés (un par côté) :



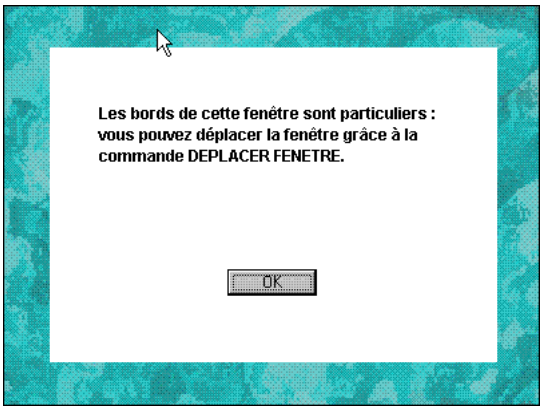
Chaque bouton est associé à la méthode suivante :

DEPLACER FENETRE ` Commencer à faire glisser la fenêtre au premier clic

Après l'exécution de la méthode projet suivante :

```
Creer fenetre(50;50;50+400;50+300;2)
DIALOGUE([Table1];"Custom Drag")
FERMER FENETRE
```

... vous obtenez une fenêtre semblable à celle-ci :



Vous pouvez la déplacer en cliquant sur les bordures.

Référence

CHANGER COORDONNEES FENETRE, COORDONNEES FENETRE.

EFFACER FENETRE

version 6.0 (Modifiée)

EFFACER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de fenêtre ou Fenêtre au premier plan du process courant si ce paramètre est omis

Description

La commande EFFACER FENETRE efface le contenu de la fenêtre dont vous avez passé la référence dans fenêtre.

Si vous omettez le paramètre fenêtre, EFFACER FENETRE efface le contenu de la fenêtre de premier plan du process courant.

Généralement, vous utiliserez EFFACER FENETRE en combinaison avec MESSAGE et POSITION MESSAGE. Dans ce cas, EFFACER FENETRE efface le contenu de la fenêtre et place le curseur dans son angle supérieur gauche, c'est-à-dire à la position correspondant à POSITION MESSAGE (0; 0).

Ne confondez pas EFFACER FENETRE, qui efface le contenu d'une fenêtre, et FERMER FENETRE, qui supprime la fenêtre de l'écran.

Référence

MESSAGE, POSITION MESSAGE.

Fenetre formulaire courant → RefFen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	RefFen	←	Numéro de référence de la fenêtre du formulaire courant

Description

La commande Fenetre formulaire courant retourne la référence de la fenêtre du formulaire courant. S'il n'y a pas de fenêtre définie pour le formulaire courant, la commande retourne 0.

La fenêtre du formulaire courant peut avoir été générée automatiquement par une commande telle que AJOUTER ENREGISTREMENT, à la suite d'une action utilisateur ou via les commandes Creer fenetre ou Creer fenetre formulaire.

Référence

Creer fenetre, Creer fenetre formulaire, REDIMENSIONNER FENETRE FORMULAIRE.

Fenetre premier plan {(*)} → RefFen

Paramètre	Type		Description
*	*	→	Si omis = ignorer les fenêtres flottantes Si spécifié = prendre en compte les fenêtres flottantes
Résultat	RefFen	←	Numéro de référence de fenêtre

Description

La commande Fenetre premier plan retourne le numéro de référence de la fenêtre actuellement située au premier plan.

Référence

Fenetre suivante, Process de premier plan.

Fenetre suivante

Fenetre suivante (fenêtre) → RefFen

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre
Résultat	RefFen	←	Numéro de référence de fenêtre

Description

La commande Fenetre suivante retourne le numéro de référence de la fenêtre située “derrière” la fenêtre dont vous avez passé le numéro de référence dans fenêtre (en fonction de l'ordre des fenêtres).

Référence

Fenetre premier plan.

FERMER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre externe ou Fenêtre de premier plan du process si ce paramètre est omis

Description

FERMER FENETRE referme la dernière fenêtre créée à l'aide de la commande Creer fenetre ou Creer fenetre formulaire dans le process courant. S'il n'y a pas de fenêtre personnalisée ouverte, FERMER FENETRE ne fait rien ; la commande ne ferme pas les fenêtres système. Si FERMER FENETRE est appelée alors qu'un formulaire est actif dans la fenêtre, elle n'a pas d'effet non plus. Vous devez appeler FERMER FENETRE lorsque vous avez fini d'utiliser une fenêtre ouverte avec Creer fenetre ou Creer fenetre formulaire.

Il est inutile de passer un numéro à FERMER FENETRE lorsque vous l'utilisez pour refermer des fenêtres ouvertes à l'aide de la fonction Creer fenetre ou Creer fenetre formulaire. En effet, si plusieurs fenêtres ont été ouvertes par une succession d'appels à ces commandes, elles ne pourront être refermées que dans l'ordre inverse de leur création.

Si vous passez en paramètre la référence d'une zone externe créée à l'aide de la fonction Creer fenetre externe, FERMER FENETRE referme la fenêtre externe. Pour plus d'informations sur les fenêtres externes, reportez-vous à la description de la fonction Creer fenetre externe.

Exemple

L'exemple suivant ouvre une fenêtre et crée des enregistrements à l'aide de la commande AJOUTER ENREGISTREMENT. Une fois les enregistrements ajoutés, la fenêtre est fermée par la commande FERMER FENETRE :

```
Creer fenetre (5; 40; 250; 300; 0; "Nouvel employé")
Repeter
   AJOUTER ENREGISTREMENT ([Employés]) ` Ajout d'un enregistrement d'employé
Jusque (OK = 0) ` Boucle jusqu'à ce que l'utilisateur annule
FERMER FENETRE ` Fermeture de la fenêtre
```

Référence

Creer fenetre, Creer fenetre externe, Creer fenetre formulaire.

LISTE FENETRES

version 6.0

LISTE FENETRES (fenêtres{; *})

Paramètre	Type		Description
fenêtres	Tableau	←	Tableau des numéros de référence des fenêtres
*	*	→	Si omis, ignorer fenêtres flottantes Si spécifié, tenir compte des fenêtres flottantes

Description

La commande LISTE FENETRES remplit le tableau fenêtres avec les numéros de référence des fenêtres actuellement ouvertes dans tous les process (process moteur et process utilisateur).

Si vous ne passez pas le paramètre optionnel *, les fenêtres flottantes sont ignorées.

Exemple

La méthode projet suivantes place en "mosaïque" toutes les fenêtres ouvertes (à l'exception des fenêtres flottantes et des boîtes de dialogue) :

```
      ` Méthode projet Mosaïque

LISTE FENETRES($alWnd)
$vlLeft:=10
$vlTop:=80 ` Laissons de la place à la barre d'outils
Boucle ($vlWnd;1;Taille tableau($alWnd))
   Si (Type fenetre($alWnd{$vlWnd}) # Fenêtre modale)
      COORDONNEES FENETRE($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlWR:=$vlLeft+($vlWR-$vlWL)
      $vlWB:=$vlTop+($vlWB-$vlWT)
      $vlWL:=$vlLeft
      $vlWT:=$vlTop
      CHANGER COORDONNEES FENETRE($vlWL;$vlWT;$vlWR;$vlWB;
                                                  $alWnd{$vlWnd})
      $vlLeft:=$vlLeft+10
      $vlTop:=$vlTop+25
   Fin de si
Fin de boucle
```

Note : Cette méthode pourrait être améliorée par l'ajout de tests sur la taille de la fenêtre principale (sous Windows) ou sur la taille et l'emplacement du ou des écran(s) (sous Mac OS).

Référence

Process de la fenetre, Type fenetre.

MAXIMISER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre Si omis = Toutes les fenêtres (Windows) ou Fenêtre de premier plan du process courant (Mac OS)

Description

La commande MAXIMISER FENETRE provoque le zoom de la fenêtre dont vous avez passé le numéro de référence dans fenêtre ou, si ce paramètre est omis, de toutes les fenêtres de l'application (sous Windows) ou de la fenêtre de premier plan du process courant (sous Mac OS).

Cette commande produit le même effet qu'un clic sur la case de zoom d'une fenêtre de l'application 4D :

Sous Windows
La fenêtre est agrandie et s'adapte à la taille courante de la fenêtre de l'application. Si vous ne passez pas le paramètre fenêtre, toutes les fenêtres de l'application sont maximisées. La fenêtre maximisée est passée au premier plan.


Case de zoom ("bouton d'agrandissement") sous Windows

Sous Mac OS
La fenêtre est agrandie de manière à afficher la totalité de son contenu. Si vous ne passez pas le paramètre fenêtre, la fenêtre du premier plan du process courant est maximisée.


Case de zoom sous Mac OS

A noter que :
• Les fenêtres que vous souhaitez maximiser doivent comporter une case de zoom. Si le type de fenêtre n'en contient pas, la commande ne fait rien (pour plus d'informations sur ce point, reportez-vous à la section Types de fenêtres).
• Sous Mac OS, le zoom étant calculé par rapport au contenu de la fenêtre, cette commande doit être appelée dans un contexte où ce contenu est défini, par exemple une méthode formulaire. Sinon, la commande ne fait rien.
• Si fenêtre est déjà maximisée, la commande ne fait rien.

La fenêtre est dimensionnée à sa taille "maximale". Si la fenêtre est un formulaire dont la taille maximale a été définie dans les Propriétés du formulaire, le zoom de la fenêtre se limitera à cette taille.

Un clic ultérieur sur la case de zoom ou l'appel de la commande MINIMISER FENETRE provoque le retour de la fenêtre à sa taille initiale.
Sous Windows, un clic sur la case de zoom ou l'appel de la commande MINIMISER FENETRE (sans paramètre) entraîne le retour à leur taille initiale de toutes les fenêtres de l'application.

Exemple

Vous souhaitez que votre formulaire s'ouvre sur une fenêtre "plein écran". Pour cela, vous placez la commande MAXIMISER FENETRE dans la méthode du formulaire :

```
` Méthode formulaire

MAXIMISER FENETRE
```

Référence

MINIMISER FENETRE.

MINIMISER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre Si omis = Toutes les fenêtres (Windows) ou Fenêtre de premier plan du process courant (Mac OS)

Description

La commande MINIMISER FENETRE provoque un zoom arrière de la fenêtre dont vous avez passé le numéro de référence dans fenêtre ou, si ce paramètre est omis, de toutes les fenêtres de l'application (sous Windows) ou de la fenêtre de premier plan du process courant (sous Mac OS).

Cette commande produit le même effet qu'un clic sur la case de réduction d'une fenêtre de l'application 4D ayant été préalablement maximisée :

Sous Windows
La fenêtre est réduite et retrouve sa taille initiale. Si vous ne passez pas le paramètre fenêtre, toutes les fenêtres de l'application sont redimensionnées à leur taille initiale.

Case de réduction sous Windows

Sous Mac OS
La fenêtre est réduite et retrouve sa taille initiale. Si vous ne passez pas le paramètre fenêtre, la fenêtre de premier plan du process courant est minimisée.

Case de zoom/réduction sous Mac OS

Si la ou les fenêtres concernées n'ont pas été préalablement maximisées (manuellement ou à l'aide de MAXIMISER FENETRE), la commande ne fait rien. De même, si le type de fenêtre ne comporte pas de case de zoom, la commande ne fait rien (pour plus d'informations sur ce point, reportez-vous à la section Types de fenêtres).

Note : Ne confondez pas cette fonction avec la réduction de la fenêtre sous forme d'icône (Windows) ou dans le Dock (Mac OS), accessible par l'intermédiaire du bouton suivant :

Windows

Mac OS

Référence

MAXIMISER FENETRE.

Process de la fenetre

Process de la fenetre {(fenêtre)} → Numérique

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de fenêtre
Résultat	Numérique	←	Numéro de référence de process

Description

La commande Process de la fenetre retourne le numéro du process qui exécute la fenêtre dont le numéro de référence est passé dans fenêtre. Si la fenêtre n'existe pas, la commande retourne 0 (zéro).

Si vous omettez le paramètre fenêtre, Process de la fenetre retourne le numéro du process de la fenêtre de premier plan du process courant.

Référence

Numero du process courant.

REDESSINER FENETRE

REDESSINER FENETRE {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande REDESSINER FENETRE provoque une mise à jour du contenu de la fenêtre dont le numéro de référence est passé dans fenêtre.

Si vous omettez le paramètre fenêtre, REDESSINER FENETRE s'appliquera à la fenêtre de premier plan du process courant.

Note : 4D gère automatiquement les mises à jour graphiques des fenêtres à chaque fois que vous déplacez, redimensionnez ou passez au premier plan une fenêtre ainsi qu'à chaque fois que vous changez le formulaire et/ou les valeurs affiché(e)s dans une fenêtre. Cette commande est rarement utilisée.

Référence

EFFACER FENETRE.

REDIMENSIONNER FENETRE FORMULAIRE

REDIMENSIONNER FENETRE FORMULAIRE (largeur; hauteur)

Paramètre	Type		Description
largeur	Entier long	→	Pixels à ajouter ou soustraire à la largeur courante de la fenêtre formulaire
hauteur	Entier long	→	Pixels à ajouter ou soustraire à la hauteur courante de la fenêtre formulaire

Description

La commande REDIMENSIONNER FENETRE FORMULAIRE permet de modifier la taille de la fenêtre du formulaire courant.

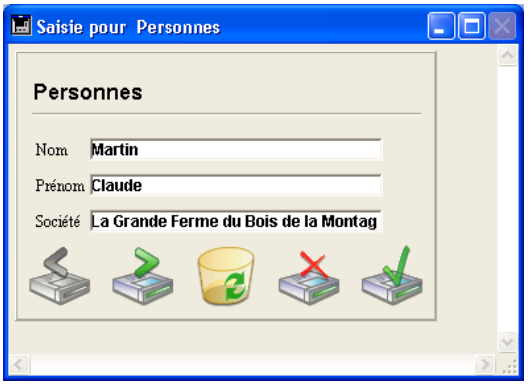
Passez dans les paramètres largeur et hauteur le nombre de pixels que vous souhaitez ajouter aux dimensions courantes de la fenêtre. Pour ne pas modifier une dimension, passez 0 dans le paramètre correspondant. Pour réduire une dimension, passez une valeur négative dans largeur et hauteur.

Cette commande produit exactement le même résultat qu'un redimensionnement manuel de la fenêtre à l'aide de la case de redimensionnement (si le type de fenêtre le permet). Par conséquent, la commande tient compte des propriétés de redimensionnement des objets et des contraintes de taille définies dans les propriétés du formulaire : si par exemple la commande entraîne un redimensionnement de la fenêtre supérieur à la taille maximale du formulaire, elle n'a pas d'effet.

A noter que ce fonctionnement est différent de celui de la commande CHANGER COORDONNEES FENETRE, qui ne tient pas compte des propriétés du formulaire ni de son contenu en cas de redimensionnement de la fenêtre.
A noter également que cette commande ne modifie pas forcément les dimensions du formulaire lui-même. Pour modifier par programmation la taille d'un formulaire, reportez-vous à la description de la commande FIXER TAILLE FORMULAIRE.

Exemple

Soit la fenêtre suivante (les champs et l'encadrement ont pour propriété de redimensionnement horizontal “Agrandir”) :



Après l'exécution de cette ligne :

REDIMENSIONNER FENETRE FORMULAIRE(25;0)

... la fenêtre a l'apparence suivante :

Référence

CHANGER COORDONNEES FENETRE, FIXER TAILLE FORMULAIRE, LIRE PROPRIETES FORMULAIRE.

Titre fenetre {(fenêtre)} → Chaîne

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis
Résultat	Chaîne	←	Titre de la fenêtre

Description

La commande Titre fenetre retourne le titre de la fenêtre dont le numéro de référence est passé dans fenêtre. Si la fenêtre n'existe pas, une chaîne vide est retournée.

Si vous omettez le paramètre fenêtre, Titre fenetre retourne le titre de la fenêtre de premier plan du process courant.

Exemple

Reportez-vous à l'exemple de la commande CHANGER TITRE FENETRE.

Référence

CHANGER TITRE FENETRE.

Type fenetre {(fenêtre)}

Paramètre	Type		Description
fenêtre	RefFen	→	Numéro de référence de la fenêtre ou Fenêtre de premier plan du process courant si omis

Description

La commande Type fenetre retourne le type de fenêtre 4D dont vous avez passé la référence dans fenêtre. Si la fenêtre n'existe pas, Type fenetre retourne 0 (zéro).

Sinon, Type fenetre retourne une des constantes prédéfinies suivantes :

Constante	Type	Valeur
Fenêtre standard	Entier long	8
Fenêtre modale	Entier long	9
Fenêtre externe	Entier long	5
Fenêtre flottante	Entier long	14

Si vous omettez le paramètre fenêtre, Type fenetre s'applique à la fenêtre de premier plan du process courant.

Exemple

Reportez-vous à l'exemple de la commande LISTE FENETRES.

Référence

COORDONNEES FENETRE, Process de la fenetre, Titre fenetre.

21

Fonctions mathématiques

Note préliminaire
Si vous ne faites pas de développement multi-plate-forme, vous pouvez tout simplement ignorer cette section.

Dans le monde de l'informatique, l'arithmétique des nombres décimaux tient plus de la technologie que de la science exacte. Vous avez avez appris en classe, par exemple, qu'un tiers peut être écrit comme une suite infinie de 3 après la virgule. Un ordinateur, quant à lui, ne le sait pas mais doit cependant évaluer cette expression. De la même manière, vous savez que trois tiers sont égal à un ; l'ordinateur calculera l'expression pour obtenir le résultat. Suivant le type d'ordinateur que vous utilisez, un tiers sera évalué comme un nombre fini de trois après la virgule. Ce nombre est appelé la précision de la machine.

Sur les anciens Macintosh à base de processeur 68K, ce nombre est égal à 19 ; cela signifie qu'un tiers sera évalué avec 19 chiffres significatifs. Sous Windows ou sur une machine à base de PowerPC, ce chiffre est égal à 15 ; cela signifie qu'un tiers sera évalué avec 15 chiffres significatifs. Si vous affichez la valeur 1/3 dans le débogueur de 4D, vous obtiendrez 0,3333333333333333333 sur les Macintosh à base de processeur 68K et environ 0,3333333333333333148 sous Windows ou sur une machine à base de processeur PowerPC. Il est à noter que les trois derniers chiffres sont différents car la précision des Macintosh à base de 68K est supérieure à celle des PC ou des machines à base de PowerPC. Cependant, si vous affichez l'expression (1/3)*3 vous obtenez 1 sur tous ces types de machines.

Si vos calculs décimaux se rapportent au nombre de mètres carrés de votre jardin, il est probable que vous n'aurez qu'un besoin très relatif des chiffres après la virgule. D'un autre côté, si vous remplissez une déclaration d'impôts, vous pouvez à certains moments vous sentir plus concerné par la précision de votre ordinateur. Cependant, une précision de 15 ou 19 est suffisante, même pour gérer des milliards de francs de revenus !

Pourquoi la valeur 1/3 semble-t-elle différente entre des Macintosh à base de processeur 68K et des machines Windows ou à base de PowerPC ?

Sur les Macintosh à base de processeur 68K, le système d'exploitation stocke les nombres réels sur 10 octets (80 bits), alors que sur un PC ou un Power Macintosh les nombres réels sont stockés sur 8 octets (64 bits). C'est pour cette raison que les nombres réels ont jusqu'à 19 chiffres significatifs sur des Macintosh à base de processeur 68K et jusqu'à 15 chiffres significatifs sur PC et Power Macintosh.

Pourquoi l'expression (1/3)*3 retourne-t-elle 1 sur ces trois types de machines ?

Un ordinateur ne peut faire que des calculs approximatifs. Par conséquent, lorsqu'il compare ou calcule des nombres, il ne traite pas les nombres réels comme des éléments mathématiques mais comme des valeurs approximatives. Dans l'exemple évoqué plus haut, 0,3333... multiplié par 3 est égal à 0,9999..., cette valeur est tellement proche de 1 que la machine considère que le résultat est égal à 1. Pour plus d'informations sur ce point, reportez vous à la commande FIXER NIVEAU COMPARAISON REEL.

Une distinction doit être établie entre :
• La manière dont les nombres réels sont calculés et comparés,
• La manière dont les nombres réels sont affichés à l'écran (ou imprimés).

A l'origine, 4D manipulait les nombres réels à l'aide du type standard de 10 octets correspondant au système d'exploitation des Macintosh 68K. De ce fait, les valeurs réelles sauvegardées sur disque sont sauvegardées dans ce format. Afin de maintenir une compatibilité entre toutes les versions (Windows, Power Macintosh et Macintosh) de 4D, les fichiers de données utilisent toujours le même type de format de 10 octets. Comme les calculs internes sont réalisés sur 8 octets pour les versions PC et Power Macintosh, 4D convertit en interne les valeurs de 10 à 8 octets, ou inversement. De ce fait, si vous chargez un enregistrement sous Windows ou sur une machine à base de PowerPC, et que cet enregistrement contient des valeurs réelles qui ont été sauvegardées sur une machine à base de processeur 68K, il est possible de perdre de la précision (en passant de 19 à 15 chiffres significatifs). Si vous chargez un enregistrement sur un Macintosh à base de processeur 68K et que cet enregistrement contient des valeurs réelles créées ou stockées sur une machine Windows ou à base de processeur PowerPC, il n'y aura aucune perte de précision. De manière générale, si vous utilisez une base de données sur des machines à base de processeur 68K, PowerPC ou une machine Windows, comptez sur une précision de 15 et non 19 chiffres.

A l'aide de la commande FIXER PARAMETRE BASE, vous pouvez définir le nombre de chiffres qui doivent être ignorés pour l'affichage des nombres réels à l'écran (4 par défaut).

Abs (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre dont vous voulez obtenir la valeur absolue
Résultat	Numérique	←	Valeur absolue de nombre

Description

Abs retourne la valeur absolue (positive et sans signe) de nombre. Si nombre est négatif, sa valeur positive est retournée. Si nombre est positif, il est retourné inchangé.

Exemple

L'exemple suivant retourne la valeur absolue de -10,3, qui est 10,3 :

```
vVector := Abs (-10,3)
```

Arctan (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Tangente pour laquelle vous souhaitez calculer l'angle en radians
Résultat	Numérique	←	Angle en radians

Description

Arctan retourne en radians la valeur de l'angle dont la tangente est spécifiée par nombre.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Exemple

Cet exemple permet d'afficher la valeur de Pi :

```
ALERTE ("Pi est égal à : "+Chaine(Arctan(1)*4))
```

Référence

Cos, Sin, Tan.

Arrondi (nombre; nbDécimales) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre à arrondir
nbDécimales	Numérique	→	Nombre de décimales de l'arrondi
Résultat	Numérique	←	Valeur de nombre arrondie avec une précision égale à nbDécimales

Description

Arrondi retourne la valeur arrondie de nombre avec une précision égale à nbDécimales.

Si nbDécimales est positif, l'arrondi se fait sur la partie décimale de nombre. Si nbDécimales est négatif, l'arrondi se fait sur la partie entière de nombre.

Si le chiffre placé derrière le nombre de décimales défini par nbDécimales est compris entre 5 et 9, nombre est arrondi à la valeur supérieure s'il est positif et inférieure s'il est négatif. Si le chiffre placé derrière la dernière décimale est compris entre 0 et 4, la fonction arrondit nombre vers zéro.

Exemple

L'exemple suivant illustre la manière dont Arrondi fonctionne dans différents cas. A chaque ligne, une valeur est assignée à la variable vRésultat. Les commentaires décrivent le résultat :

```
vRésultat := Arrondi (16,857; 2)   ` vRésultat vaut 16,86
vRésultat := Arrondi (32345,67; -3) ` vRésultat vaut 32000
vRésultat := Arrondi (29,8725; 3)  ` vRésultat vaut 29,873
vRésultat := Arrondi (-1,5; 0)   ` vRésultat vaut -2
```

Référence

Troncature.

Convertisseur Euro

Convertisseur Euro (valeur; deMonnaie; versMonnaie) → Numérique

Paramètre	Type		Description
valeur	Numérique	→	Valeur à convertir
deMonnaie	Alpha	→	Code ISO de la monnaie dans laquelle la valeur est exprimée
versMonnaie	Alpha	→	Code ISO de la monnaie dans laquelle la valeur doit être convertie
Résultat	Numérique	←	Valeur convertie

Description

La commande Convertisseur Euro vous permet d'effectuer tout type de conversion de valeurs entre les différentes monnaies des pays de la "zone euro" et l'Euro lui-même.

Vous pouvez convertir :
• une monnaie nationale en Euros,
• des Euros en une monnaie nationale,
• une monnaie nationale en une autre monnaie nationale. Dans ce cas, la conversion s'effectue toujours par l'intermédiaire de l'Euro, comme le stipule la réglementation. Par exemple, pour convertir des Francs belges en Marks allemands, 4D effectuera les conversions suivantes : Francs belges -> Euro -> Marks allemands.

Vous passez dans le premier paramètre la valeur à convertir.
Dans le second paramètre, vous indiquez le code ISO de la monnaie dans laquelle valeur est exprimée.
Dans le troisième paramètre, vous indiquez le code ISO de la monnaie dans laquelle vous souhaitez que valeur soit convertie.

Pour désigner les codes ISO, 4D vous propose les constantes prédéfinies suivantes, placées dans le thème "Euro monnaies" :

Constante	Type	Valeur
Drachme grecque	Alpha	GRD
Escudo portugais	Alpha	PTE
Euro	Alpha	EUR
Florin néerlandais	Alpha	NLG
Franc belge	Alpha	BEF
Franc français	Alpha	FRF
Franc luxembourgeois	Alpha	LUF

Lire italienne	Alpha	ITL
Livre irlandaise	Alpha	IEP
Mark allemand	Alpha	DEM
Mark finlandais	Alpha	FIM
Peseta espagnole	Alpha	ESP
Schilling autrichien	Alpha	ATS

Si nécessaire, 4D arrondit automatiquement le résultat des conversions et conserve 2 décimales — à l'exception des conversions vers les Lires italiennes, Francs luxembourgeois, Francs belges et Pesetas espagnoles, pour lesquelles 4D conserve 0 décimale (le résultat est un nombre entier).

La parité des différentes monnaies vis-à-vis de l'Euro est fixe. Les taux de conversion, utilisés en interne par 4D, sont les suivants :

Monnaie	**Valeur pour 1 Euro**
Drachme grecque	340,750
Escudo portugais	200,482
Florin néerlandais	2,20371
Franc belge	40,3399
Franc français	6,55957
Franc luxembourgeois	40,3399
Lire italienne	1936,27
Livre irlandaise	0,787564
Mark allemand	1,95583
Mark finlandais	5,94573
Peseta espagnole	166,386
Schilling autrichien	13,7603

Exemple

Voici différents types de conversion pouvant être obtenus à l'aide de cette commande :

```
$valeur:=10000  `Valeur exprimée en francs français
   `Convertir la valeur en euros
$EnEuros:=Convertisseur Euro ($valeur;Franc français; Euro)
   `Convertir la valeur en lires italiennes
$EnLires:=Convertisseur Euro ($valeur;Franc français; Lire italienne)
```

Cos (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre, exprimé en radians, dont vous voulez connaître le cosinus
Résultat	Numérique	←	Cosinus de nombre

Description

Cos retourne le cosinus de nombre. La valeur nombre est exprimée en radians.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Référence

Arctan, Sin, Tan.

Dec (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Valeur dont voulez obtenir la partie décimale
Résultat	Numérique	←	Partie décimale de nombre

Description

Dec retourne la partie décimale de nombre. La valeur retournée est toujours positive ou nulle.

Exemple

L'exemple suivant utilise une valeur monétaire exprimée sous forme numérique et en extrait les parties "euros" et "centimes". Si vrMontant valait 7,31, vlEuros vaudrait 7 et vlCentimes 31 :

```
vlEuros := Ent (vrMontant) ` Extraire les euros
    ` Extraire la partie décimale et la multiplier par 100 pour obtenir un entier
vlCentimes := Dec (vrMontant) * 100
```

Référence

Ent.

Ent (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Valeur dont vous voulez obtenir la partie entière
Résultat	Numérique	←	Partie entière de nombre

Description

Ent retourne la partie entière de nombre en l'arrondissant à l'entier inférieur.

Exemple

L'exemple suivant illustre le fonctionnement de Ent pour les nombres positifs et négatifs. Notez que la partie décimale du nombre est supprimée :

```
x := Ent (123,4) ` x vaut 123
y := Ent (-123,4) ` y vaut –124
```

Référence

Dec.

Exp (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre à évaluer
Résultat	Numérique	←	Exponentielle de nombre

Description

Exp retourne l'exponentielle (e=2,71828...) de nombre. Exp est la fonction inverse de Log.

Note : La fonction exponentielle, qui au nombre réel x fait correspondre le nombre réel y, est notée $y = e^x$. 4D fournit la constante prédéfinie Nombre e (2,71828...).

Exemple

L'exemple suivant assigne l'exponentielle de 1 à vrE (le logarithme de vrE est 1) :

```
vrE := Exp (1)    ` vrE vaut e (e = 2,71828...)
```

Référence

Log.

FIXER NIVEAU COMPARAISON REEL (epsilon)

Paramètre	Type		Description
epsilon	Numérique	→	Valeur epsilon pour les comparaisons d'égalité des réels

Description

La commande FIXER NIVEAU COMPARAISON REEL définit la valeur epsilon utilisée par 4D lors d'une comparaison d'égalité des valeurs et expressions de type Réel.

Comme un ordinateur effectue des calculs approximatifs sur les réels, les tests sur l'égalité de valeurs réelles doivent tenir compte de cette approximation. Pour cela, 4D, lorsqu'il compare des valeurs réelles, teste en fait si la différence entre les deux valeurs est supérieure ou non à une certaine valeur. Cette valeur s'appelle l'**epsilon** et fonctionne de la manière suivante : Soient deux valeurs réelles a et b. Si Abs(a-b) est supérieur à l'epsilon, les valeurs sont considérées comme différentes ; sinon, elles sont déclarées égales.
Par défaut, 4D fixe la valeur epsilon à 10 à la puissance moins 6 (10^-6). Exemples :
• 0,00001=0,00002 retourne Faux car la différence 0,00001 est supérieure à 10^-6.
• 0,000001=0,000002 retourne Vrai car la différence 0,000001 n'est pas supérieure à 10^-6.
• 0,000001=0,000003 retourne Faux car la différence 0,000002 est supérieure à 10^-6.

A l'aide de FIXER NIVEAU COMPARAISON REEL, vous pouvez augmenter ou réduire la valeur epsilon, en fonction de vos besoins.

Note : La commande n'aura pas d'effet si epsilon > 10^-3 ou si epsilon < 0.

Modifier l'epsilon affecte seulement la comparaison d'égalité des réels. Cela n'a pas d'effet sur les calculs et l'affichage des valeurs réelles.

ATTENTION : Cette commande doit être utilisée dans des cas spécifiques, comme par exemple un tri sur un champ dont les valeurs sont inférieures à 10^-6. En général, vous n'avez pas besoin de modifier la valeur par défaut d'epsilon.

Note : Si vous souhaitez effectuer un traitement (tri, recherche) sur un champ numérique indexé dont les valeurs sont inférieures à 10^-6, veillez à ce que la commande FIXER NIVEAU COMPARAISON REEL soit exécutée **avant** la construction de l'index.

Référence
Opérateurs de comparaison.

Hasard → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Nombre aléatoire

Description

Hasard retourne une valeur entière aléatoire comprise entre 0 et 32 767 (inclus).

Pour que la valeur aléatoire soit située dans un intervalle donné, utilisez la formule suivante :

```
(Hasard % (fin – début + 1)) + début
```

La valeur début est le premier nombre de l'intervalle, fin est le dernier.

Exemple

L'exemple suivant assigne une valeur entière aléatoire entre 10 et 30 à la variable vRésultat :

```
Résultat := (Hasard % 21) + 10
```

Log (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre dont vous voulez obtenir le logarithme népérien
Résultat	Numérique	←	Logarithme népérien de nombre

Description

Log retourne le logarithme népérien de nombre. Log est la fonction inverse de Exp.

Note : 4D fournit la constante prédéfinie Nombre e (2,71828...).

Exemple

L'exemple suivant affiche 1 :

```
ALERTE (Chaine(Log(Exp (1)))
```

Référence

Exp.

Modulo (nombre1; nombre2) → Numérique

Paramètre	Type	Description
nombre1	Numérique →	Nombre à diviser (numérateur)
nombre2	Numérique →	Nombre diviseur (dénominateur)
Résultat	Numérique ←	Reste de la division entière de nombre1 par nombre2

Description

La fonction Modulo divise nombre1 par nombre2 et retourne le reste sous forme d'un nombre entier.

Notes :

• Modulo accepte des expressions de type Entier, Entier long et Réel (numérique). Cependant, si nombre1 et/ou nombre2 sont des nombres réels, ils sont arrondis avant le calcul du Modulo.
• La fonction Modulo est à utiliser avec précaution avec des nombres réels de grande taille (au-delà de 2^31). Dans ce cas en effet, son fonctionnement peut se heurter aux limites des capacités de calcul des processeurs standard.

Vous pouvez également utiliser l'opérateur "%" pour calculer le reste d'une division (reportez-vous à la section Opérateurs numériques). Toutefois, cet opérateur retourne des résultats valides uniquement avec des expressions de type Entier et Entier long. Si vous voulez calculer le modulo de nombres réels, vous devez utiliser la commande Modulo.

Exemple

L'exemple suivant illustre le fonctionnement de Modulo dans différents cas de figure. A chaque ligne, un nombre est assigné à la variable vRésultat. Les commentaires fournissent le résultat obtenu :

```
vRésultat := Modulo(3;2)   ` vRésultat prend la valeur 1
vRésultat := Modulo(4;2)   ` vRésultat prend la valeur 0
vRésultat := Modulo(3,5;2)   ` vRésultat prend la valeur 0
```

Référence

Opérateurs numériques.

Racine carree (nombre) → Numérique

Paramètre	Type	Description
nombre	Numérique →	Nombre duquel calculer la racine carrée
Résultat	Numérique ←	Racine carrée de nombre

Description

Racine carree retourne la racine carrée de nombre.

Exemples

(1) La ligne :

```
$vrRacineDeDeux := Racine carree (2)
```

affecte la valeur 1,414213562373 à la variable $vrRacineDeDeux.

(2) La méthode listée ci-dessous retourne l'hypoténuse du triangle rectangle dont les deux côtés sont passés en paramètres :

```
` Méthode Hypoténuse
` Hypoténuse ( Réel ; Réel ) -> Réel
` Hypoténuse ( côtéA ; côtéB ) -> Hypoténuse
C_REEL($0;$1;$2)
$0 := Racine carree(($1^2)+($2^2))
```

Par exemple, Hypoténuse (4;3) retourne 5.

Référence

Opérateurs numériques.

Sin (nombre) → Numérique

Paramètre	Type	Description
nombre	Numérique →	Nombre, exprimé en radians, dont vous voulez connaître le sinus
Résultat	Numérique ←	Sinus de nombre

Description

Sin retourne le sinus de nombre. La valeur nombre est exprimée en radians.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Référence

Arctan, Cos, Tan.

Tan (nombre) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre, exprimé en radians, dont vous voulez connaître la tangente
Résultat	Numérique	←	Tangente de nombre

Description

Tan retourne la tangente de nombre. La valeur nombre est exprimée en radians.

Note : 4D fournit les constantes prédéfinies Pi, Degré et Radian. Pi retourne le nombre Pi (3,14159...), Degré retourne la valeur en radians d'un degré (0,01745...) et Radian retourne la valeur en degrés d'un radian (57,29577...).

Référence

Arctan, Cos, Sin.

Troncature (nombre; nbDécimales) → Numérique

Paramètre	Type		Description
nombre	Numérique	→	Nombre à tronquer
nbDécimales	Numérique	→	Nombre de décimales à conserver
Résultat	Numérique	←	nombre tronqué à partir du nombre de décimales indiqué par nbDécimales

Description

Troncature retourne nombre dont la partie décimale a été tronquée à partir du nombre de décimales spécifié par nbDécimales. Troncature arrondit toujours nombre à la valeur inférieure.

Si nbDécimales est positif, la troncature se fait sur la partie décimale de nombre. Si nbDécimales est négatif, la troncature se fait sur la partie entière de nombre.

Exemple

L'exemple suivant illustre la manière dont Troncature fonctionne dans différents cas. A chaque ligne, une valeur est assignée à la variable vRésultat. Les commentaires décrivent le résultat :

```
vRésultat := Troncature (216,897; 1)    ` Résultat prend la valeur 216,8
vRésultat := Troncature (216,897; -1)   ` Résultat prend la valeur 210
vRésultat := Troncature (-216,897; 1)   ` Résultat prend la valeur -216,9
vRésultat := Troncature (-216,897; -1)  ` Résultat prend la valeur -220
```

Référence

Arrondi.

22

Fonctions statistiques

Les fonctions de ce thème effectuent des calculs sur des séries de valeurs.

Les fonctions Moyenne, Max, Min, Somme, Somme des carres, Ecart type et Variance s'appliquent aux champs. Elles utilisent une sélection d'enregistrements. Notez que les fonctions Somme des carres, Ecart type et Variance ne peuvent être utilisées que pendant l'impression.

Les fonctions ne travaillent que sur des valeurs numériques. Chacune de ces fonctions retourne une valeur numérique.

Utilisation des fonctions statistiques hors impression
Lorsque les fonctions Moyenne, Max, Min ou Somme sont utilisées sur un champ en-dehors d'une opération d'impression, il se peut qu'elles aient à charger chaque enregistrement de la sélection courante pour calculer le résultat. S'il y a beaucoup d'enregistrements, l'opération peut être longue. Pour limiter la durée du traitement, vous pouvez indexer le champ.

Note : Lorsque l'opération est longue, un thermomètre de progression apparaît. Ce thermomètre comporte un bouton **Arrêt** permettant à l'utilisateur d'interrompre l'opération. Si l'utilisateur clique sur ce bouton, la variable OK prend la valeur 0. Si l'opération est correctemen menée à son terme, la variable OK prend la valeur 1.

Utilisation des fonctions statistiques dans un état imprimé
Lorsque les fonctions statistiques sont utilisées dans un état, elles se comportent de façon particulière, car c'est l'état lui-même qui charge chaque enregistrement. Utilisez ces fonctions dans une méthode formulaire ou objet quand vous imprimez avec la commande IMPRIMER SELECTION ou quand vous imprimez en mode Développement à l'aide de la commande **Imprimer** dans le menu **Fichier**.
Lorsque vous utilisez ces fonctions dans un état, les valeurs retournées sont fiables uniquement dans le niveau 0 de rupture et seulement si la phase de traitement des ruptures est active. Cela signifie qu'elles ne sont utiles qu'en fin d'état, lorsque tous les enregistrements ont été traités.
Vous ne devez utiliser ces fonctions avec une méthode objet que dans une zone non-saisissable incluse dans la zone de rupture R0.
Pour mémoire : un champ passé comme paramètre à une fonction statistique doit être un numérique.

Référence

Ecart type, Max, Min, Moyenne, Somme, Somme des carres, Variance.

Ecart type (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous voulez obtenir l'écart type
Résultat	Numérique	←	Ecart type de séries

Description

Ecart type retourne l'écart type (c.-à-d. la racine carrée de la variance) de séries. Si séries est un champ indexé, l'index sera utilisé pour le calcul.

Exemple

L'exemple suivant est la méthode objet d'une variable appelée EcartT. La méthode assigne l'écart type d'une série à EcartT :

```
EcartT := Ecart type ([Table1]SérieValeurs)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection et activer la phase de rupture :

```
TOUT SELECTIONNER ([Table1])
TRIER ([Table1];[Table1]SérieValeurs;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Table1]SérieValeurs)
FORMULAIRE SORTIE ([Table1];"FormImpression")
IMPRIMER SELECTION ([Table1])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

Référence

Moyenne, Somme, Somme des carres, Variance.

Max (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous voulez obtenir la valeur la plus élevée
Résultat	Numérique	←	Valeur la plus élevée de séries

Description

Max retourne la valeur la plus élevée contenue dans séries. Si séries est un champ indexé, l'index est utilisé pour la recherche de cette valeur.

Si la sélection de séries est vide, Max retourne -1E50.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple

L'exemple ci-dessous est la méthode objet d'une variable, vMax, placée dans la zone de rupture R0 d'un formulaire sortie. La variable est imprimée à la fin de l'état. La méthode objet assigne la valeur la plus élevée du champ à la variable, qui est alors imprimée dans la dernière rupture de l'état.

```
vMax := Max ([Employés] Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer la phase de rupture :

```
TOUT SELECTIONNER ([Employés])
TRIER ([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Employés]Salaire)
FORMULAIRE SORTIE ([Employés];"FormImpression")
IMPRIMER SELECTION ([Employés])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

Référence

Min.

Min (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous voulez obtenir la valeur la plus basse
Résultat	Numérique	←	Valeur la plus basse de séries

Description

Min retourne la valeur la plus faible de séries. Si séries est un champ indexé, l'index est utilisé pour la recherche de cette valeur.

Si la sélection de séries est vide, Min retourne 1E50.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemples

(1) L'exemple ci-dessous est la méthode objet d'une variable, vMin, placée dans la zone de rupture R0 d'un formulaire sortie. La variable est imprimée à la fin de l'état. La méthode objet assigne la valeur la plus basse du champ à la variable, qui est alors imprimée dans la dernière rupture de l'état.

```
vMin := Min ([Employés] Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer la phase de rupture :

```
TOUT SELECTIONNER ([Employés])
TRIER ([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Employés]Salaire)
FORMULAIRE SORTIE ([Employés];"FormImpression")
IMPRIMER SELECTION ([Employés])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

(2) L'exemple suivant recherche la plus petite vente d'un employé et affiche le résultat dans une boîte de dialogue d'alerte :

```
ALERTE ("Plus petite vente = " + Chaine(Min([Employés]Ventes)))
```

Référence

Max.

Moyenne (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous voulez calculer la moyenne
Résultat	Numérique	←	Moyenne arithmétique de séries

Description

Moyenne retourne la moyenne arithmétique de séries. Si séries est un champ indexé, l'index est utilisé pour le calcul.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple

Dans l'exemple suivant, une valeur est assignée à une variable se trouvant dans la zone de rupture R0 d'un formulaire sortie. La ligne de code ci-dessous constitue la méthode objet de la variable. Elle n'est exécutée qu'à l'impression du niveau de rupture 0 :

```
vMoyenne := Moyenne ([Employés]Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer la phase de rupture :

```
TOUT SELECTIONNER ([Employés])
TRIER ([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Employés]Salaire)
FORMULAIRE SORTIE ([Employés];"FormImpression")
IMPRIMER SELECTION ([Employés])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

Référence

CUMULER SUR, IMPRIMER SELECTION, Max, Min, NIVEAUX DE RUPTURES, Somme, Sous total, TRIER.

Somme (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous souhaitez calculer la somme
Résultat	Numérique	←	Somme de séries

Description

Somme retourne la somme (c'est-à-dire le total de toutes les valeurs) de séries. Si séries est un champ indexé, l'index est utilisé pour le calcul.

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Si elle est interrompue (par exemple si l'utilisateur clique sur le bouton **Arrêt** dans le thermomètre de progression), la variable OK prend la valeur 0.

Exemple

L'exemple ci-dessous est la méthode objet d'une variable, vTotal, placée dans un formulaire. La méthode assigne à la variable la somme de tous les salaires :

```
vTotal := Somme ([Employés]Salaire)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection courante et activer le traitement des ruptures :

```
TOUT SELECTIONNER ([Employés])
TRIER ([Employés];[Employés]Nom;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Employés]Salaire)
FORMULAIRE SORTIE ([Employés];"FormImpression")
IMPRIMER SELECTION ([Employés])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

Référence

CUMULER SUR, IMPRIMER SELECTION, Max, Min, Moyenne, NIVEAUX DE RUPTURES, Sous total, TRIER.

Somme des carres (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous voulez obtenir la somme des carrés
Résultat	Numérique	←	Somme des carrés de séries

Description

Somme des carres retourne la somme des carrés de séries. Si séries est un champ indexé, l'index est utilisé pour le calcul.

Exemple

L'exemple suivant est la méthode objet d'une variable appelée Carrés. La méthode assigne la somme des carrés d'une série de valeurs à Carrés. La méthode est imprimée dans la dernière rupture de l'état :

```
Carrés := Somme des carres ([Table1]SérieValeurs)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection et activer la phase de rupture :

```
TOUT SELECTIONNER ([Table1])
TRIER ([Table1];[Table1]SérieValeurs;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Table1]SérieValeurs)
FORMULAIRE SORTIE ([Table1];"FormImpression")
IMPRIMER SELECTION ([Table1])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

Référence

Ecart type, Moyenne, Somme, Variance.

Variance (séries) → Numérique

Paramètre	Type		Description
séries	Champ	→	Champ dont vous voulez obtenir la variance
Résultat	Numérique	←	Variance de séries

Description

Variance retourne la variance de séries. Si séries est un champ indexé, l'index est utilisé pour le calcul.

La variance d'un ensemble de valeurs est la moyenne des carrés des écarts par rapport à la moyenne. C'est une valeur de dispersion autour de la moyenne. 4D utilise la formule de variance suivante :

Variance(x) = Somme (x-m)*(x-m)/(n-1)
m = Moyenne
n = Nombre de valeurs

Si les valeurs considérées ne constituent pas un échantillon, multipliez la valeur retournée par Variance par (n-1)/n.

Exemple

L'exemple suivant est la méthode objet d'une variable appelée Var. La méthode assigne la variance de la série à Var:

```
Var:= Variance ([Etudiants]Notes)
```

La méthode suivante est appelée pour imprimer les enregistrements de la sélection et activer la phase de rupture :

```
TOUT SELECTIONNER ([Etudiants])
TRIER ([Etudiants];[Etudiants]Classe;>)
NIVEAUX DE RUPTURES (1)
CUMULER SUR ([Etudiants]Notes)
FORMULAIRE SORTIE ([Etudiants];"FormImpression")
IMPRIMER SELECTION ([Etudiants])
```

Note : La valeur du paramètre de la commande NIVEAUX DE RUPTURES doit être égale au nombre de ruptures que contient l'état. Pour plus d'informations sur les ruptures, reportez-vous aux commandes du thème Impressions.

Référence

Ecart type, Moyenne, Somme, Somme des carres.

23

Formulaires

ALLER A PAGE

version 2004.3 (Modifiée)

ALLER A PAGE (numéroPage)

Paramètre	Type		Description
numéroPage	Numérique	→	Numéro de la page à afficher

Description

ALLER A PAGE change la page courante d'un formulaire pour afficher la page désignée par numéroPage.

Si aucun formulaire n'est affiché ou si numéroPage correspond à la page courante du formulaire, ALLER A PAGE ne fait rien. Si numéroPage est supérieur au nombre de pages du formulaire, la dernière page est affichée. Si numéroPage est inférieur à 1, la première page est affichée.

A propos des commandes de gestion des pages formulaire

4D vous fournit des actions automatiques pour les boutons qui effectuent les mêmes tâches que les commandes PREMIERE PAGE, DERNIERE PAGE, PAGE SUIVANTE, PAGE PRECEDENTE, ALLER A PAGE que vous pouvez associer aux objets tels que les onglets, les listes déroulantes, etc. A chaque fois que c'est possible, utilisez les actions automatiques pour les boutons plutôt que ces commandes.

Les commandes de gestion des pages peuvent être utilisées avec des formulaires entrée ou des formulaires affichés dans des boîtes de dialogue. Les formulaires sortie n'utilisent que la première page. Un formulaire comprend toujours au minimum une page, la première. Notez bien que quel que soit le nombre de pages qu'il contient, un formulaire ne peut être associé qu'à une seule méthode formulaire.

Vous pouvez utiliser la commande Page formulaire courante pour savoir quelle page est affichée à l'écran.

Note : Pendant que vous construisez un formulaire, vous pouvez utiliser les pages 1 à N du formulaire ainsi que la page 0 (zéro), dans laquelle vous placez les objets que vous voulez faire apparaître sur toutes les pages. Lors de l'utilisation du formulaire, et donc lorsque les commandes de gestion des pages sont appelées, seules les pages 1 à N sont accessibles : la page 0 est automatiquement combinée à la page affichée à l'écran.

Exemple

L'exemple suivant est la méthode objet d'un bouton affichant la page 3 du formulaire :

```
ALLER A PAGE (3)
```

Référence

DERNIERE PAGE, Page formulaire courante, PAGE PRECEDENTE, PAGE SUIVANTE, PREMIERE PAGE.

DERNIERE PAGE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande DERNIERE PAGE change la page courante d'un formulaire pour afficher la dernière page du formulaire. Si aucun formulaire n'est affiché ou si la dernière page du formulaire est déjà affichée, DERNIERE PAGE ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle affiche la dernière page du formulaire courant :

```
DERNIERE PAGE
```

Référence

ALLER A PAGE, Page formulaire courante, PAGE PRECEDENTE, PAGE SUIVANTE, PREMIERE PAGE.

FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL (redimension{; largeurMini{; largeurMaxi}})

Paramètre	Type		Description
redimension	Booléen	→	Vrai : le formulaire est redimensionnable horizontalement Faux : le formulaire n'est pas redimensionnable horizontalement
largeurMini	Entier long	→	Largeur minimale du formulaire (pixels)
largeurMaxi	Entier long	→	Largeur maximale du formulaire (pixels)

Description

La commande FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL permet de modifier par programmation les propriétés de redimensionnement horizontal du formulaire courant. Par défaut, ces propriétés sont définies dans l'éditeur de formulaires en mode Développement. Les nouvelles propriétés sont fixées pour le process courant, elles ne sont pas stockées avec le formulaire.

Le paramètre redimension permet de définir si le formulaire est redimensionnable horizontalement, c'est-à-dire si sa largeur est modifiable (manuellement par l'utilisateur ou par programmation).
Si vous passez Vrai, la largeur du formulaire peut être modifiée par l'utilisateur ; 4D utilise comme bornes les valeurs éventuellement passées dans les paramètres largeurMini et largeurMaxi.
Si vous passez Faux, la largeur courante du formulaire n'est pas modifiable ; dans ce cas, il est inutile de passer des valeurs dans les paramètres largeurMini et largeurMaxi.

Si vous avez passé Vrai dans le premier paramètre, vous pouvez passer dans les paramètres facultatifs largeurMini et largeurMaxi les nouvelles largeurs minimale et maximale du formulaire en pixels. Si vous omettez ces paramètres, les valeurs définies en mode Développement (le cas échéant) seront utilisées.

Exemple

Reportez-vous à l'exemple de la commande FIXER TAILLE FORMULAIRE.

Référence

FIXER REDIMENSIONNEMENT FORMULAIRE VERTICAL, FIXER TAILLE FORMULAIRE.

FIXER REDIMENSIONNEMENT FORMULAIRE VERTICAL (redimension{; hauteurMini{; hauteurMaxi}})

Paramètre	Type		Description
redimension	Booléen	→	Vrai : le formulaire est redimensionnable verticalement Faux : le formulaire n'est pas redimensionnable verticalement
hauteurMini	Entier long	→	Hauteur minimale du formulaire (pixels)
hauteurMaxi	Entier long	→	Hauteur maximale du formulaire (pixels)

Description

La commande FIXER REDIMENSIONNEMENT FORMULAIRE VERTICAL permet de modifier par programmation les propriétés de redimensionnement vertical du formulaire courant. Par défaut, ces propriétés sont définies dans l'éditeur de formulaires en mode Développement. Les nouvelles propriétés sont fixées pour le process courant, elles ne sont pas stockées avec le formulaire.

Le paramètre redimension permet de définir si le formulaire est redimensionnable verticalement, c'est-à-dire si sa hauteur est modifiable (manuellement par l'utilisateur ou par programmation).
Si vous passez Vrai, la hauteur du formulaire peut être modifiée par l'utilisateur ; 4D utilise comme bornes les valeurs éventuellement passées dans les paramètres hauteurMini et hauteurMaxi.
Si vous passez Faux, la hauteur courante du formulaire n'est pas modifiable ; dans ce cas, il est inutile de passer des valeurs dans les paramètres hauteurMini et hauteurMaxi.

Si vous avez passé Vrai dans le premier paramètre, vous pouvez passer dans les paramètres facultatifs hauteurMini et hauteurMaxi les nouvelles hauteurs minimale et maximale du formulaire en pixels. Si vous omettez ces paramètres, les valeurs définies en mode Développement (le cas échéant) seront utilisées.

Exemple

Reportez-vous à l'exemple de la commande FIXER TAILLE FORMULAIRE.

Référence

FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL, FIXER TAILLE FORMULAIRE.

FIXER TAILLE FORMULAIRE ({objet; }horizontal; vertical{; *})

Paramètre	Type		Description
objet	Chaîne	→	Nom d'objet indiquant les limites du formulaire
horizontal	Entier long	→	Si * passé : marge horizontale (pixels) Si * omis : largeur (pixels)
vertical	Entier long	→	Si * passé : marge verticale (pixels) Si * omis : hauteur (pixels)
*	*	→	• Si passé, ajouter les marges définies par les paramètres horizontal et vertical (taille automatique ou taille basée sur objet, si objet est passé) • Si omis, utiliser horizontal et vertical comme largeur et hauteur du formulaire

Description

La commande FIXER TAILLE FORMULAIRE permet de modifier par programmation la taille du formulaire courant. La nouvelle taille est définie pour le process courant, elle n'est pas stockée avec le formulaire.

Comme en mode Développement, cette commande permet de définir la taille d'un formulaire de trois manières :
• automatiquement — 4D détermine la taille du formulaire sur le principe que tous les objets doivent être visibles — en ajoutant éventuellement une marge horizontale et une marge verticale,
• sur la base de l'emplacement d'un objet du formulaire auquel s'ajoutent éventuellement une marge horizontale et une marge verticale,
• en saisissant des dimensions "absolues" (largeur et hauteur).
Pour plus d'informations sur les possibilités de dimensionnement des formulaires, reportez-vous au manuel *Mode Développement* de 4D.

• Taille automatique
Pour que le formulaire ait une taille automatique, vous devez utiliser la syntaxe suivante :

FIXER TAILLE FORMULAIRE(horizontal; vertical;*)

Dans ce cas, vous devez passer dans horizontal et vertical les marges (en pixels) que vous souhaitez ajouter à droite et en bas du formulaire.

• Taille basée sur un objet
Pour que la taille du formulaire soit basée sur un objet, vous devez utiliser la syntaxe suivante :

FIXER TAILLE FORMULAIRE(objet; horizontal; vertical;*)

Dans ce cas, vous devez passer dans horizontal et vertical les marges (en pixels) que vous souhaitez ajouter à droite et en bas de l'objet.

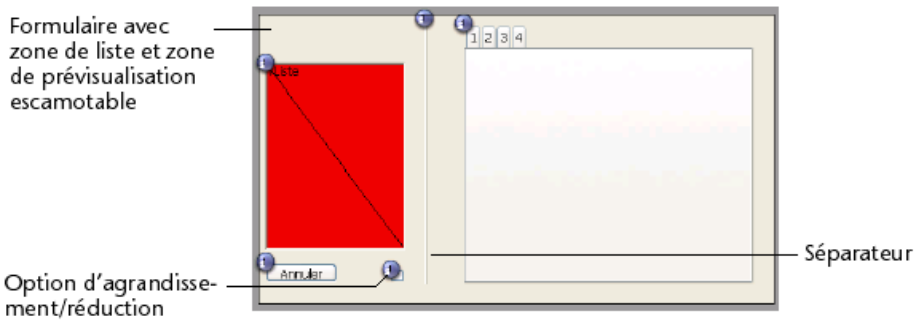
• Taille en valeur absolue
Pour passer une taille de formulaire absolue, vous devez utiliser la syntaxe suivante :

FIXER TAILLE FORMULAIRE(horizontal; vertical)

Dans ce cas, vous devez passer dans horizontal et vertical la largeur et la hauteur (en pixels) du formulaire.

La commande FIXER TAILLE FORMULAIRE modifie la taille du formulaire mais tient compte de ses propriétés de redimensionnement. Par exemple, si la largeur minimale du formulaire est de 500 pixels et si la commande définit une largeur de 400 pixels, la nouvelle largeur du formulaire sera de 500 pixels.
A noter également que cette commande ne modifie pas la taille de la fenêtre du formulaire (il est possible de redimensionner un formulaire sans que la taille de la fenêtre soit modifiée, et inversement). Pour modifier la taille de la fenêtre d'un formulaire, reportez-vous à la description de la commande REDIMENSIONNER FENETRE FORMULAIRE.

Exemple
Voici un exemple de mise en place d'une fenêtre de type Explorateur. Le formulaire suivant est défini en mode Développement :

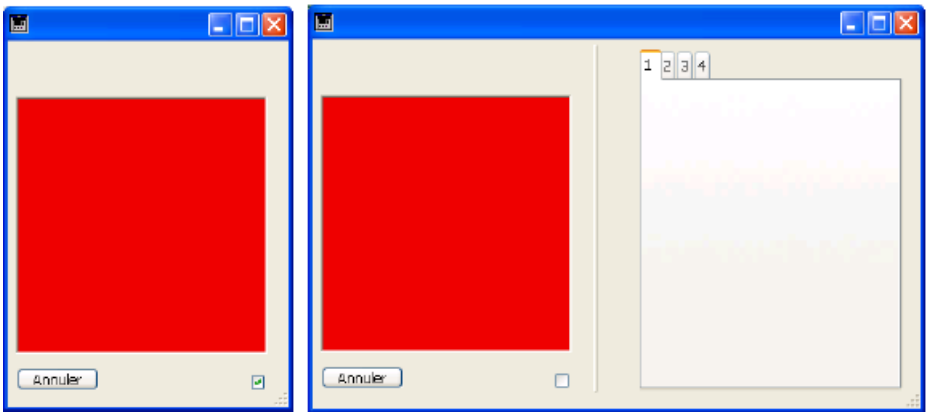


La taille du formulaire est "automatique".

La fenêtre est affichée via l'instruction suivante :

```
$ref:=Creer fenetre formulaire([Table 1];"Form1";Form fenêtre standard;
                                        Centrée horizontalement;Centrée verticalement ;*)
DIALOGUE([Table 1];"Form1")
FERMER FENETRE
```

La partie droite de la fenêtre peut être affichée ou masquée via un clic sur l'option d'agrandissement/réduction :



La méthode objet associée à ce bouton est la suivante :

```
Au cas ou
: (Evenement formulaire=Sur chargement )
    C_BOOLEEN(b1;<>contracté)
    C_ENTIER LONG(marge)
    marge:=15
    b1:=<>contracté
    Si (<>contracté)
        FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL(Faux)
        FIXER TAILLE FORMULAIRE("b1";marge;marge)
    Sinon
        FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL(Vrai)
        FIXER TAILLE FORMULAIRE("onglet";marge;marge)
    Fin de si

: (Evenement formulaire=Sur clic)
    <>contracté:=b1
    Si (b1)
            `contracté
        LIRE RECT OBJET(*;"b1";$g;$h;$d;$b)
                COORDONNEES FENETRE($gf;$hf;$df;$bf;Fenetre formulaire
```

```
            CHANGER COORDONNEES FENETRE($gf;$hf;$gf+$d+marge;$hf+$b+marge;
                                                    Fenetre formulaire courant)
            FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL(Faux)
            FIXER TAILLE FORMULAIRE("b1";marge;marge)

        Sinon
            `déployé
            LIRE RECT OBJET(*;"onglet";$g;$h;$d;$b)
            COORDONNEES FENETRE($gf;$hf;$df;$bf;Fenetre formulaire courant)
            CHANGER COORDONNEES FENETRE($gf;$hf;$gf+$d+marge;$hf+$b+marge;
                                                    Fenetre formulaire courant)
            FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL(Vrai)
            FIXER TAILLE FORMULAIRE("onglet";marge;marge)
        Fin de si

    Fin de cas
```

Référence

FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL, FIXER REDIMENSIONNEMENT FORMULAIRE VERTICAL.

FORMULAIRE ENTREE ({laTable; }formulaire{; formUtilisateur}{; *})

Paramètre	Type		Description
laTable	Table	→	Table pour laquelle définir le formulaire entrée ou Table par défaut si ce paramètre est omis
formulaire	Alpha	→	Nom du formulaire à utiliser
formUtilisateur	Alpha	→	Nom du formulaire utilisateur à utiliser
*	*	→	Taille de fenêtre automatique

Description

FORMULAIRE ENTREE désigne formulaire ou formUtilisateur comme formulaire entrée courant de laTable pour le process courant. formulaire doit appartenir à laTable.

La portée de cette commande est le process courant. Chaque table dispose d'un formulaire entrée courant pour chaque process.

Note : Pour des raisons structurelles, cette commande n'est pas compatible avec les formulaires projets. Si vous passez un formulaire projet dans formulaire, la commande ne fait rien.

La commande FORMULAIRE ENTREE n'affiche pas de formulaire ; elle désigne juste celui qui sera affiché ou utilisé par d'autres commandes. Pour plus d'informations sur la création de formulaires, reportez-vous au manuel *Mode Développement* de 4D.

Pour chaque table de la base, un formulaire entrée par défaut est défini dans la fenêtre de l'Explorateur. Ce formulaire est utilisé si la commande FORMULAIRE ENTREE n'est pas appelée, ou si le paramètre formulaire est invalide.

Le paramètre facultatif formUtilisateur permet de désigner un formulaire utilisateur (issu du formulaire) comme formulaire entrée par défaut. Si vous passez un nom de formulaire utilisateur valide, ce formulaire sera utilisé par défaut en lieu et place du formulaire entrée dans le process courant. Ce principe permet de disposer simultanément de différents formulaires utilisateurs personnalisés (générés à l'aide de la commande CREER FORMULAIRE UTILISATEUR) et d'utiliser celui qui convient en fonction du contexte.
Pour plus d'informations sur les formulaires utilisateurs, reportez-vous à la section Présentation des formulaires utilisateurs.

Le formulaire entrée est affiché par de nombreuses commandes. Ces commandes sont généralement utilisées pour la saisie ou la modification de valeurs. Les commandes suivantes affichent un formulaire entrée :

- AJOUTER ENREGISTREMENT
- CHERCHER PAR EXEMPLE
- AFFICHER ENREGISTREMENT
- MODIFIER ENREGISTREMENT

Les commandes VISUALISER SELECTION et MODIFIER SELECTION affichent une liste d'enregistrements dans le formulaire sortie. Chacune d'entre elles permet ensuite à l'utilisateur de double-cliquer sur un enregistrement, qui s'affiche alors dans le formulaire entrée.

Le formulaire entrée est aussi utilisé par les commandes d'import LECTURE DIF, LECTURE SYLK et IMPORTER TEXTE.

Le paramètre optionnel * est destiné à être utilisé conjointement avec les propriétés du formulaire, que vous définissez en mode Développement dans la fenêtre des Propriétés du formulaire, et avec la commande Creer fenetre. En passant le paramètre *, vous indiquez à 4D d'utiliser les propriétés du formulaire pour redimensionner automatiquement la fenêtre lors de l'utilisation ultérieure de la fenêtre comme formulaire entrée ou comme dialogue. Reportez-vous à la description de la commande Creer fenetre pour plus d'informations sur ce point.

Note : Que vous passiez ou non le paramètre *, FORMULAIRE ENTREE change le formulaire entrée pour la table.

Exemples

(1) L'exemple suivant illustre une utilisation typique de FORMULAIRE ENTREE. A noter que, si dans cet exemple FORMULAIRE ENTREE est appelé juste avant que le formulaire soit utilisé, cela n'est absolument pas nécessaire. FORMULAIRE ENTREE peut en fait être exécuté dans une tout autre méthode, du moment qu'elle est exécutée avant celle-ci :

```
FORMULAIRE ENTREE ([Sociétés]; "Nouvelle Sté") ` Formulaire pour les nouvelles sociétés
AJOUTER ENREGISTREMENT ([Sociétés]) ` Ajout d'une nouvelle société
```

(2) Dans une base de facturation gérant plusieurs sociétés, la création d'une facture doit s'effectuer dans le formulaire utilisateur correspondant :

```
Au cas ou
: (société="4D SAS")
    FORMULAIRE ENTREE([Factures];"Saisie";"4D_SAS")
: (société="4D Inc")
        FORMULAIRE ENTREE([Factures];"Saisie";"4D_Inc")
```

```
: (société="Acme")
    FORMULAIRE ENTREE([Factures];"Saisie";"ACME")
Fin de cas
AJOUTER ENREGISTREMENT([Factures])
```

Référence

AFFICHER ENREGISTREMENT, AJOUTER ENREGISTREMENT, CHERCHER PAR EXEMPLE, Creer fenetre, CREER FORMULAIRE UTILISATEUR, FORMULAIRE SORTIE, IMPORTER TEXTE, LECTURE DIF, LECTURE SYLK, MODIFIER ENREGISTREMENT, MODIFIER SELECTION, VISUALISER SELECTION.

FORMULAIRE SORTIE ({laTable; }formulaire{; formUtilisateur})

Paramètre	Type		Description
laTable	Table	→	Table pour laquelle définir le formulaire sortie ou Table par défaut si ce paramètre est omis
formulaire	Alpha	→	Nom du formulaire
formUtilisateur	Alpha	→	Nom du formulaire utilisateur à utiliser

Description

FORMULAIRE SORTIE vous permet de définir formulaire ou formUtilisateur comme formulaire sortie courant de laTable pour le process courant. formulaire doit appartenir à laTable.

La portée de cette commande est le process courant. Chaque table dispose de son propre formulaire sortie dans chaque process.

Note : Pour des raisons structurelles, cette commande n'est pas compatible avec les formulaires projets. Si vous passez un formulaire projet dans formulaire, la commande ne fait rien.

La commande FORMULAIRE SORTIE ne provoque pas l'affichage du formulaire ; elle désigne simplement le formulaire devant être imprimé, affiché, ou utilisé par d'autres commandes. Pour plus d'informations sur la création de formulaires, reportez-vous au manuel *Mode Développement* de 4D.

Le formulaire sortie par défaut est défini dans la fenêtre de l'Explorateur pour chaque table. Il est identifié par la lettre S placée près de son nom dans l'Explorateur et dans les boîtes de dialogue listant les formulaires. Ce formulaire par défaut sera utilisé si vous n'appelez pas la commande FORMULAIRE SORTIE ou si vous passez à cette commande un nom de formulaire erroné ou inexistant.

Le paramètre facultatif formUtilisateur permet de désigner un formulaire utilisateur (issu du formulaire) comme formulaire sortie par défaut. Si vous passez un nom de formulaire utilisateur valide, ce formulaire sera utilisé par défaut en lieu et place du formulaire sortie dans le process courant. Ce principe permet de disposer simultanément de différents formulaires utilisateurs personnalisés (générés à l'aide de la commande CREER FORMULAIRE UTILISATEUR) et d'utiliser celui qui convient en fonction du contexte.

Pour plus d'informations sur les formulaires utilisateurs, reportez-vous à la section Présentation des formulaires utilisateurs.

Les formulaires sortie sont exploités par trois groupes de commandes. Le premier groupe gère l'affichage des enregistrements à l'écran, le deuxième gère la génération d'états et le troisième gère l'export de données.

Chacune des commandes suivantes affiche une liste d'enregistrements dans un formulaire sortie :
• VISUALISER SELECTION
• MODIFIER SELECTION

Vous utilisez le formulaire sortie lorsque vous créez des états à l'aide des commandes suivantes :
• IMPRIMER ETIQUETTES
• IMPRIMER SELECTION

Chacune des commandes d'export suivantes utilise également le formulaire sortie :
• ECRITURE DIF
• ECRITURE SYLK
• EXPORTER TEXTE

Exemple

L'exemple suivant illustre une utilisation typique FORMULAIRE SORTIE. Notez que, bien que dans cet exemple la commande FORMULAIRE SORTIE soit placée juste avant que le formulaire soit utilisé, cela n'est pas obligatoire. En fait, la commande pourrait se trouver dans n'importe quelle autre méthode, dans la mesure où elle est exécutée avant celle-ci :

```
FORMULAIRE ENTREE ([Parties]; "Saisie Parties") ` Sélection du formulaire entrée
FORMULAIRE SORTIE ([Parties]; "Liste Parties") ` Sélection du formulaire sortie
MODIFIER SELECTION ([Parties]) ` Cette commande utilise les deux formulaires
```

Référence

CREER FORMULAIRE UTILISATEUR, ECRITURE DIF, ECRITURE SYLK, EXPORTER TEXTE, FORMULAIRE ENTREE, IMPRIMER ETIQUETTES, IMPRIMER SELECTION, MODIFIER SELECTION, VISUALISER SELECTION.

LIRE OBJETS FORMULAIRE (tabObjets{; tabVariables{; tabPages}}{; *})

Paramètre	Type		Description
tabObjets	Tableau alpha	←	Noms des objets du formulaire
tabVariables	Tableau pointeur	←	Pointeurs sur les variables ou champs associés aux objets
tabPages	Tableau entier	←	Numéro de page de chaque objet
*	*	→	Si passé = réduire à la page courante

Description

La commande LIRE OBJETS FORMULAIRE retourne sous forme de tableau(x) la liste de tous les objets présents dans le formulaire courant. Cette liste peut être restreinte à la page courante du formulaire. La commande peut être utilisée avec les formulaires entrée et sortie.

Si un tableau passé en paramètre n'est pas préalablement déclaré, la commande le crée et le dimensionne automatiquement. Toutefois, dans la perspective de la compilation de l'application, il est recommandé de déclarer explicitement chaque tableau.

Passez dans tabObjets le nom du tableau alpha devant être rempli avec les noms des objets (chaque nom d'objet est unique au sein d'un formulaire). L'ordre dans lequel les objets apparaissent dans le tableau n'est pas significatif.

Les autres tableaux remplis facultativement par la commande sont synchronisés avec le premier.

Passez dans le paramètre facultatif tabVariables le nom du tableau de pointeurs devant être rempli avec des pointeurs vers les variables ou champs associés aux objets. Si un objet n'a pas de variable associée, le pointeur Nil est retourné. Dans le cas d'un objet de type "sous-formulaire", un pointeur sur la table du sous-formulaire est retourné.

Le troisième tableau (facultatif), tabPages, est rempli avec les numéros de pages du formulaire. Chaque ligne de ce tableau contient le numéro de la page sur laquelle se trouve l'objet correspondant.
Les objets provenant d'un formulaire hérité sont considérés comme appartenant à la page 0 du formulaire courant.

Le paramètre facultatif * permet de réduire la liste des objets retournés à la page courante du formulaire. Lorsque ce paramètre est passé, seuls les objets de la page courante, de la page 0 et des pages héritées sont retournés par la commande. Autrement dit, tous les objets présents dans la page courante du formulaire (visibles ou non) sont traités par la commande.

Référence

LIRE PROPRIETES FORMULAIRE.

LIRE PARAMETRE FORMULAIRE

LIRE PARAMETRE FORMULAIRE ({table; }formulaire; sélecteur; valeur)

Paramètre	Type		Description
table	Table	→	Table du formulaire ou Table par défaut si ce paramètre est omis
formulaire	Chaîne	→	Nom du formulaire
sélecteur	Entier long	→	Code du paramètre
valeur	Entier long	←	Valeur courante du paramètre

Description

La commande LIRE PARAMETRE FORMULAIRE permet de lire la valeur courante d'un paramètre du formulaire désigné par table et formulaire.

sélecteur désigne le paramètre du formulaire dont vous souhaitez connaître la valeur. Vous pouvez utiliser la constante suivante, placée dans le thème "Paramètres de formulaire" :

Constante	Type	Valeur
Objets non inversés	Entier long	0

Lorsque vous utilisez la constante Objets non inversés comme sélecteur, la commande retourne dans valeur le mode d'affichage réel du formulaire en mode Application sous Windows. Ce paramètre est utilisé dans le cadre du déploiement d'applications dans des langues "de droite à gauche". Pour plus d'informations sur la prise en charge des langues de droite à gauche, reportez-vous manuel *Mode Développement* de 4D.
- si valeur contient 0, les objets du formulaire sont inversés,
- si valeur contient 1, les objets du formulaire ne sont pas inversés.

Si la commande est appelée en-dehors du contexte du mode Application sous Windows, elle retourne toujours 1.

A noter que l'inversion effective des objets d'un formulaire dépend de la combinaison de plusieurs paramètres : la valeur de la préférence "Inversion des objets en mode Application", la valeur de l'option de formulaire "Ne pas inverser les objets" et le système sur lequel la base est exécutée.

Le tableau suivant précise la valeur retournée par la commande LIRE PARAMETRE FORMULAIRE en fonction de ces différentes combinaisons :

Préférences : "Inversion des objets en mode Application" (1)	Propriétés du formulaire : "Ne pas inverser les objets"	Système Windows Droite à gauche	Valeur retournée dans LIRE PARAMETRE FORMULAIRE
Non	X	X	1
		X	1
	X		1
			1
Automatique	X	X	1
		X	**0**
	X		1
			1
Oui	X	X	1
		X	**0**
	X		1
			0

(1) Cette préférence peut également être fixée ou lue via les commandes FIXER PARAMETRE BASE et Lire parametre base.

Référence

FIXER PARAMETRE BASE, Lire parametre base, LIRE PROPRIETES FORMULAIRE.

LIRE PROPRIETES FORMULAIRE ({table; }nomForm; largeur; hauteur{; nbPages{; largeurFixe{; hauteurFixe{; titre}}}})

Paramètre	Type		Description
table	Table	→	Table du formulaire ou Table par défaut si ce paramètre est omis
nomForm	Alpha	→	Nom du formulaire
largeur	Entier long	←	Largeur du formulaire (en pixels)
hauteur	Entier long	←	Hauteur du formulaire (en pixels)
nbPages	Entier long	←	Nombre de pages du formulaire
largeurFixe	Booléen	←	Vrai = Largeur fixe, Faux = Largeur variable
hauteurFixe	Booléen	←	Vrai = Hauteur fixe, Faux = Hauteur variable
titre	Texte	←	Nom de la fenêtre du formulaire

Description

La commande LIRE PROPRIETES FORMULAIRE retourne des propriétés du formulaire nomForm.

Les paramètres largeur et hauteur retournent (en pixels) la largeur et la hauteur du formulaire. Ces valeurs sont déterminées à partir des propriétés de dimensionnement du formulaire :

• Si la taille du formulaire est **automatique**, sa largeur et sa hauteur sont calculées de manière à ce qu'il affiche tous les objets qu'il contient, en tenant compte, le cas échéant, des marges horizontale et verticale qui ont été définies.

• Si la taille du formulaire est **fixe**, sa largeur et sa hauteur sont celles qui ont été saisies manuellement dans les zones correspondantes.

• Si la taille du formulaire est **basée sur un objet**, sa largeur et sa hauteur sont calculées par rapport à la position de cet objet.

Le paramètre nbPages retourne le nombre de pages du formulaire, page 0 (zéro) non comprise.

Les paramètres largeurFixe et hauteurFixe indiquent si la largeur et la hauteur du formulaire sont fixes (le paramètre contient Vrai) ou redimensionnables (le paramètre contient Faux).

Le paramètre titre retourne le nom de la fenêtre du formulaire, tel qu'il a été défini. Si aucun nom n'a été défini, le paramètre titre contient une chaîne vide.

Référence

Creer fenetre formulaire, FIXER TAILLE FORMULAIRE, LIRE OBJETS FORMULAIRE.

Page formulaire courante → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Numéro de la page courante du formulaire actuellement affiché

Description

Page formulaire courante retourne le numéro de la page courante du formulaire actuellement affiché.

Exemple

Alors que vous être en train d'utiliser un formulaire, si vous choisissez une commande de menu ou si le formulaire reçoit un appel d'un autre process, vous voulez que des actions différentes soient effectuées en fonction de la page du formulaire affichée. Vous pouvez alors écrire :

```
      ` Méthode formulaire [maTable];"monFormulaire"
Au cas ou
   : (Evenement formulaire=Sur chargement)
      ` ...
   : (Evenement formulaire=Sur libération)
      ` ...
   : (Evenement formulaire=Sur menu sélectionné)
      $vlNuméroMenu:=Menu choisi >> 16
      $vlNuméroCmde:=Menu choisi & 0xFFFF
      Au cas ou
         : ($vlNuméroMenu=...)
            Au cas ou
               : ($vlNuméroCmde=...)
                  : (Page formulaire courante=1)
                     ` Effectuer une action appropriée pour la page 1
```

```
                        : (Page formulaire courante=2)
                            ` Effectuer une action appropriée pour la page 2
                            ` ...
                    : ($vlNuméroCmde=...)
                            ` ...
                Fin de cas
            : ($vlNuméroMenu=...)
                ` ...
        Fin de cas
    : (Evenement formulaire=Sur appel extérieur)
        Au cas ou
            : (Page formulaire courante=1)
                ` Fournir une réponse appropriée pour la page 1
            : (Page formulaire courante=2)
                ` Fournir une réponse appropriée pour la page 2
        Fin de cas
            ` ...
Fin de cas
```

Référence

ALLER A PAGE, DERNIERE PAGE, PAGE PRECEDENTE, PAGE SUIVANTE, PREMIERE PAGE.

PAGE PRECEDENTE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

PAGE PRECEDENTE change la page courante d'un formulaire pour afficher la page précédente. Si aucun formulaire n'est affiché, ou si la page affichée est la première page du formulaire, PAGE PRECEDENTE ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle provoque l'affichage la page qui précède celle qui est actuellement affichée :

```
PAGE PRECEDENTE
```

Référence

ALLER A PAGE, DERNIERE PAGE, Page formulaire courante, PAGE SUIVANTE, PREMIERE PAGE.

PAGE SUIVANTE

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

PAGE SUIVANTE change la page courante d'un formulaire pour afficher la page suivante. Si aucun formulaire n'est affiché, ou si la page affichée est la dernière page du formulaire, PAGE SUIVANTE ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle provoque l'affichage la page qui suit celle qui est actuellement affichée :

```
PAGE SUIVANTE
```

Référence

ALLER A PAGE, DERNIERE PAGE, Page formulaire courante, PAGE PRECEDENTE, PREMIERE PAGE.

PREMIERE PAGE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande PREMIERE PAGE change la page courante d'un formulaire pour afficher la première page du formulaire. Si aucun formulaire n'est affiché, ou si la première page du formulaire est déjà affichée, PREMIERE PAGE ne fait rien.

Exemple

Cet exemple est une méthode appelée par une commande de menu. Elle affiche la première page du formulaire :

```
PREMIERE PAGE
```

Référence

ALLER A PAGE, DERNIERE PAGE, Page formulaire courante, PAGE PRECEDENTE, PAGE SUIVANTE.

24

Formulaires utilisateurs

Dans 4D, le développeur peut proposer aux utilisateurs de créer ou de modifier des formulaires personnalisés. Ces “formulaires utilisateurs” sont alors utilisables pour l'affichage, la saisie, etc., comme n’importe quel formulaire de 4D.

Principes d'utilisation

Un formulaire utilisateur est basé sur un formulaire 4D standard créé par le développeur en mode Développement (appelé formulaire “source” ou formulaire “développeur”), auquel la propriété **Modifiable par l’utilisateur** a été appliquée dans l'éditeur de formulaires. Un éditeur de formulaires simplifié (appelé par la commande MODIFIER FORMULAIRE) permet aux utilisateurs de modifier l’apparence du formulaire, d’ajouter des objets graphiques (via une bibliothèque d'objets spécifique), de masquer des éléments, etc. — le développeur peut contrôler les actions autorisées.

Les formulaires utilisateurs peuvent être employés de deux manières différentes :
• L’utilisateur modifie le formulaire “source” pour l’adapter à ses besoins à l’aide de la commande MODIFIER FORMULAIRE. Le formulaire utilisateur est conservé en local et est utilisé automatiquement à la place du formulaire original.
Ce fonctionnement répond aux besoins pour le développeur de paramétrer sur site des boîtes de dialogue, par exemple pour coller le logo de l’entreprise dans les formulaires, masquer des champs inutiles, etc.
• Le formulaire “source” sert de modèle de base que l’utilisateur peut dupliquer à loisir pour générer autant de copies que nécessaire (via la commande CREER FORMULAIRE UTILISATEUR). Chaque copie est librement paramétrable (contenu, nom, etc.) à l’aide de la commande MODIFIER FORMULAIRE. Le nom de chaque formulaire utilisateur doit simplement être unique. Les commandes FORMULAIRE ENTREE et FORMULAIRE SORTIE permettent ensuite de désigner le formulaire utilisateur à utiliser dans chaque process.
Ce fonctionnement répond par exemple aux besoins de génération d’états personnalisés.

Stockage et mise à jour des formulaires utilisateurs

Les mécanismes des formulaires utilisateurs fonctionnent avec les bases compilées et interprétées, avec 4D en mode local, 4D Server ou 4D Desktop. En mode client/serveur, les formulaires modifiés par l’utilisateur sont disponibles sur tous les postes.

4D assure automatiquement la gestion des modifications des formulaires. Lorsqu’un formulaire est déclaré **Modifiable par l’utilisateur**, il est verrouillé en mode Développement. Le développeur doit explicitement cliquer sur l’icône de déverrouillage afin de pouvoir accéder aux objets du formulaire. Cette opération rend automatiquement obsolètes les formulaires utilisateurs liés, qui devront alors être regénérés. Lorsqu’un formulaire “source” est supprimé, les formulaires utilisateurs liés sont supprimés.

Les formulaires utilisateurs sont stockés dans un fichier indépendant suffixé .4DA, placé à côté du fichier de structure principal (.4DB / .4DC). Ce fichier est appelé “fichier de structure utilisateur”. Le fonctionnement de ce fichier est transparent : 4D utilise un formulaire utilisateur lorsqu’il existe (la commande LISTE FORMULAIRES UTILISATEURS permet de connaître à tout moment les formulaires utilisateurs valides). C'est également dans ce fichier que les commandes FORMULAIRE ENTREE et FORMULAIRE SORTIE recherchent les formulaires utilisateurs.
Lorsqu’un formulaire utilisateur est obsolète, il est supprimé et 4D utilise par défaut le formulaire source.
En client/serveur, le fichier .4DA est distribué sur les postes clients suivant les mêmes règles que le fichier de structure principal.
Ce principe permet de conserver les formulaires utilisateurs non obsolètes en cas de mise à jour de la structure par le développeur.

Codes d'erreurs

Des codes d'erreurs spécifiques peuvent être retournés lors de l'utilisation des commandes de gestion des formulaires utilisateurs. Ces codes, situés dans l'intervalle -9750 à -9759, sont décrits dans la section Erreurs de la base de données.

Formulaires utilisateurs et formulaires projet

Les mécanismes des formulaires utilisateurs ne sont pas compatibles avec les formulaires projet. Les commandes du thème “Formulaires utilisateurs” ne peuvent donc pas être utilisées avec les formulaires projet.

CREER FORMULAIRE UTILISATEUR (table; formulaire; formUtilisateur)

Paramètre	Type		Description
table	Table	→	Table du formulaire source
formulaire	Chaîne	→	Nom du formulaire table source
formUtilisateur	Chaîne	→	Nom du nouveau formulaire utilisateur

Description

La commande CREER FORMULAIRE UTILISATEUR duplique le formulaire table 4D dont la table et le nom sont passés en paramètres et crée un nouveau formulaire utilisateur nommé formUtilisateur.

Une fois créé, le formulaire formUtilisateur pourra être modifié à l'aide de la commande MODIFIER FORMULAIRE. Cette commande permet de créer N formulaires utilisateurs (par exemple divers formulaires d'états) à partir d'un même formulaire source.

Référence

FORMULAIRE ENTREE, FORMULAIRE SORTIE, LISTE FORMULAIRES UTILISATEURS, MODIFIER FORMULAIRE, Présentation des formulaires utilisateurs, SUPPRIMER FORMULAIRE UTILISATEUR.

Variables et ensembles système

La variable *OK* retourne 1 si l'opération s'est déroulée correctement et 0 sinon.

Gestion des erreurs

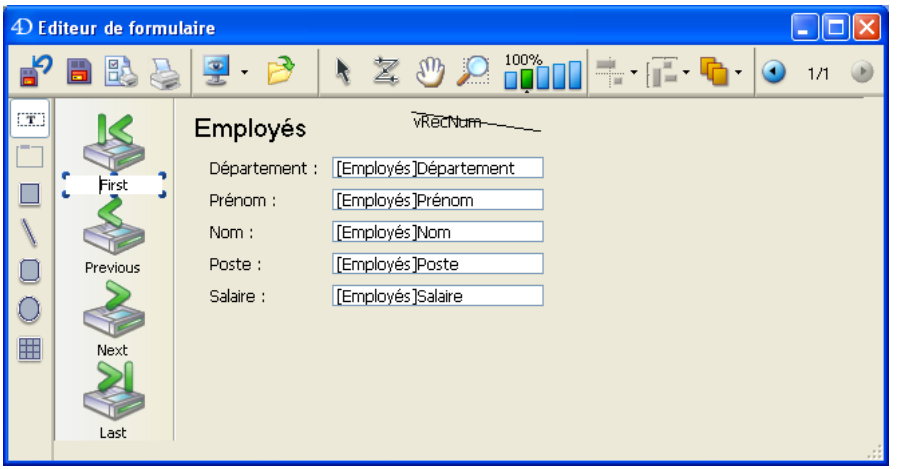
Une erreur est générée si :
• formulaire est déjà un formulaire utilisateur,
• le nom du formulaire utilisateur formUtilisateur est identique à celui du formulaire source ou d'un formulaire utilisateur existant,
• l'utilisateur ne possède pas les droits d'accès adéquats.
Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

MODIFIER FORMULAIRE (table; formulaire{; formUtilisateur{; bibliothèque}})

Paramètre	Type		Description
table	Table	→	Table du formulaire à modifier
formulaire	Chaîne	→	Nom du formulaire table à modifier
formUtilisateur	Chaîne	→	Nom du formulaire utilisateur à modifier
bibliothèque	Chaîne	→	Chemin d'accès complet de la bibliothèque d'objets utilisable

Description

La commande MODIFIER FORMULAIRE ouvre le formulaire table désigné par les paramètres table, formulaire ainsi que, facultativement, formUtilisateur dans l'éditeur de formulaires utilisateurs :



Note : La fenêtre de l'éditeur ne s'ouvre que si elle est la première fenêtre du process. Autrement dit, il sera généralement nécessaire d'ouvrir un nouveau process pour afficher l'éditeur.

Si vous passez une chaîne vide dans le paramètre formUtilisateur et s'il n'existe pas déjà un formulaire utilisateur lié à formulaire, le formulaire source est affiché dans l'éditeur. Le formulaire modifié est ensuite dupliqué dans le fichier de structure utilisateur (.4DA) et sera utilisé en remplacement de formulaire.

Si un formulaire utilisateur avait déjà été généré à partir de formulaire à l'aide de cette commande, le formulaire utilisateur s'affiche dans l'éditeur. Si vous souhaitez dans ce cas repartir du formulaire source, vous devez au préalable supprimer le formulaire utilisateur à l'aide de la commande SUPPRIMER FORMULAIRE UTILISATEUR.

Le paramètre formUtilisateur permet de désigner un formulaire utilisateur (créé à l'aide de la commande CREER FORMULAIRE UTILISATEUR) à modifier. Dans ce cas, ce formulaire est affiché dans l'éditeur.

Passez dans le paramètre facultatif bibliothèque le chemin d'accès complet de la bibliothèque d'objets que l'utilisateur sera autorisé à utiliser pour personnaliser le formulaire. Lorsqu'elles sont utilisées dans le contexte de l'éditeur de formulaires utilisateurs, les bibliothèques d'objets permettent de coller des objets avec leurs propriétés graphiques et leurs actions automatiques. Les objets auxquels une méthode est associée n'apparaissent pas dans la bibliothèque. Attention, il est du ressort du développeur de vérifier que l'ajout des objets d'une bibliothèque n'est pas incompatible avec le formulaire utilisateur (et ses objets) au niveau des noms, des variables et des types.
En mode client/serveur, la bibliothèque doit se trouver dans le dossier **Resources** de la base de données, au même niveau que le dossier **Plugins**, afin qu'elle soit disponible sur tous les postes clients. Si la bibliothèque est valide, elle est ouverte avec la fenêtre du formulaire. Pour plus d'informations sur les bibliothèques d'objets, reportez-vous au manuel *Mode Développement* de la documentation de 4D.

Exemple

Dans cet exemple, l'utilisateur peut choisir une bibliothèque puis modifier un formulaire de dialogue :

```
ASSOCIER TYPES FICHIER("4DLB";"4IL";"Bibliothèque 4D")
$vAbib:=Selectionner document(1;"4DLB";"Veuillez sélectionner une bibliothèque";0)
Si(OK=1)
        `Une bibliothèque a été choisie
    $vACheminLib:=Document
Sinon
    $vACheminLib:=""
Fin de si

MODIFIER FORMULAIRE([Dialogs];"Welcome";"";$vACheminLib)
Si(OK=1)
        `Présentation du formulaire modifié
    DIALOGUE([Dialogs];"Welcome")
Fin de si
```

Référence

CREER FORMULAIRE UTILISATEUR, LISTE FORMULAIRES UTILISATEURS, Présentation des formulaires utilisateurs, SUPPRIMER FORMULAIRE UTILISATEUR.

Variables et ensembles système

Si l'utilisateur sauvegarde les modifications éventuellement effectuées dans l'éditeur, la variable *OK* prend la valeur 1. En cas d'erreur, *OK* prend la valeur 0.

Gestion des erreurs

Une erreur est générée si :
• le formulaire n'a pas été déclaré modifiable par l'utilisateur en mode Développement ou n'existe pas,
• le formulaire est déjà ouvert en modification dans un autre process,
• l'utilisateur ne possède pas les droits d'accès adéquats.
Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

LISTE FORMULAIRES UTILISATEURS (table; formulaire; tabFormUtilisateurs)

Paramètre	Type		Description
table	Table	→	Table du formulaire source
formulaire	Chaîne	→	Nom du formulaire table source
tabFormUtilisateurs	Tab Alpha	←	Noms des formulaires utilisateurs issus du formulaire source

Description

La commande LISTE FORMULAIRES UTILISATEURS remplit le tableau tabFormUtilisateurs avec les noms des formulaires utilisateurs issus du formulaire développeur (formulaire table) désigné par les paramètres table et formulaire.

Si le formulaire utilisateur a été créé directement à l'aide de la commande MODIFIER FORMULAIRE, tabFormUtilisateurs contient comme seul élément une chaîne vide ("").

Le tableau est retourné vide si aucun formulaire utilisateur n'existe pour le formulaire développeur spécifié.

Référence

CREER FORMULAIRE UTILISATEUR, MODIFIER FORMULAIRE, Présentation des formulaires utilisateurs.

SUPPRIMER FORMULAIRE UTILISATEUR (table; formulaire; formUtilisateur)

Paramètre	Type		Description
table	Table	→	Table du formulaire utilisateur
formulaire	Chaîne	→	Nom du formulaire table source
formUtilisateur	Chaîne	→	Nom du formulaire utilisateur

Description

La commande SUPPRIMER FORMULAIRE UTILISATEUR permet de supprimer le formulaire utilisateur désigné par les paramètres table, formulaire et formUtilisateur.

Si le formulaire utilisateur à supprimer a été créé directement à l'aide la commande MODIFIER FORMULAIRE, passez une chaîne vide ("") dans formUtilisateur.

Référence

CREER FORMULAIRE UTILISATEUR, LISTE FORMULAIRES UTILISATEURS, Présentation des formulaires utilisateurs.

Variables et ensembles système

Si le formulaire utilisateur est correctement supprimé, la variable *OK* retourne 1. Dans le cas contraire, *OK* prend la valeur 0.

Gestion des erreurs

Une erreur est générée si :
• le formulaire utilisateur n'existe pas,
• l'utilisateur ne possède pas les droits d'accès adéquats.
Vous pouvez intercepter ces erreurs à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

25

Formules

EDITER FORMULE (table; formule)

Paramètre	Type		Description
table	Table	→	Table à afficher par défaut dans l'éditeur de formules
formule	Variable chaîne	→	Variable contenant la formule à afficher dans l'éditeur de formules ou "" pour uniquement afficher l'éditeur
		←	Formule validée par l'utilisateur

Description

La commande EDITER FORMULE affiche l'éditeur de formules afin de permettre à l'utilisateur d'écrire ou de modifier une formule. L'éditeur contient à l'ouverture :
• dans la liste de gauche, les champs de la table désignée par le paramètre table,
• dans la zone de formule, la formule contenue dans la variable formule. Si vous avez passé une chaîne vide dans formule, l'éditeur est affiché sans formule.

L'utilisateur peut modifier la formule affichée et la sauvegarder. Il peut également en écrire ou en charger une nouvelle. Dans tous les cas, lorsque l'utilisateur valide la boîte de dialogue, la variable système *OK* prend la valeur 1 et la variable formule contient la formule définie. Si l'utilisateur annule la boîte de dialogue, la variable système *OK* prend la valeur 0 et formule est inchangée.

Note : Par défaut, l'accès aux méthodes et aux commandes est restreint dans l'éditeur de formules pour tous les utilisateurs (sauf, dans les bases de données créées avec 4D 2004.4 et suivantes, pour le Super_Utilisateur et l'Administrateur). Lorsque ce mécanisme est actif, vous devez explicitement désigner les éléments accessibles aux utilisateurs à l'aide de la commande FIXER METHODES AUTORISEES. Si la formule fait appel à des méthodes qui n'ont pas été préalablement autorisées, une erreur de syntaxe est générée et il n'est pas possible de valider la boîte de dialogue.

A noter qu'au moment de la validation de la boîte de dialogue, la commande n'exécute pas la formule, seul le contenu de la variable est validé et mis à jour. Si vous voulez exécuter la formule, vous devez utiliser la commande EXECUTER.

Exemple

Affichage de l'éditeur avec la table [Salaires] et sans formule pré-saisie puis exécution de la formule sur la sélection courante :

```
$maFormule:=""
EDITER FORMULE([Salaires];$maFormule)
Si (OK=1)
    APPLIQUER A SELECTION([Salaires];EXECUTER FORMULE($maFormule))
Fin de si
```

Référence

APPLIQUER A SELECTION, EXECUTER, FIXER METHODES AUTORISEES.

Variables et ensembles système

Si l'utilisateur valide la boîte de dialogue, la variable système *OK* prend la valeur 1. Si l'utilisateur annule la boîte de dialogue, la variable système *OK* prend la valeur 0.

EXECUTER FORMULE (instruction)

Paramètre	Type		Description
instruction	Alpha	→	Code à exécuter

Description

EXECUTER FORMULE exécute instruction comme une ligne de code. Cette chaîne d'instructions doit comporter une seule ligne. Si instruction est une chaîne vide, EXECUTER FORMULE ne fait rien.

Le principe est que si instruction peut être exécutée comme une méthode d'une seule ligne, alors elle s'exécutera correctement. La commande EXECUTER FORMULE doit être utilisée avec précautions, car elle ralentit la vitesse d'exécution. Dans une base compilée, le code d'instruction n'est pas compilé. Cela signifie que l'instruction sera bien exécutée, mais ne sera pas vérifiée par le compilateur au moment de la compilation.

L'instruction peut contenir les éléments suivants :
• un appel à une méthode projet,
• un appel à une commande 4D,
• une assignation.

La formule peut utiliser des variables process et interprocess. En revanche, instruction ne doit pas contenir d'instructions de contrôle de flux (Si, Tant que...) car le code doit "tenir" sur une seule ligne.

Exemple

Reportez-vous à l'exemple de la commande Nom commande.

Référence

EDITER FORMULE, Nom commande.

FIXER METHODES AUTORISEES (tabMéthodes)

Paramètre	Type		Description
tabMéthodes	Tableau Alpha	→	Tableau de noms de méthodes

Description

La commande FIXER METHODES AUTORISEES permet de définir les méthodes qui seront affichées dans l'éditeur de formules pour la session courante. Les méthodes désignées apparaîtront à la fin de la liste des commandes et pourront être utilisées dans les formules. Par défaut (si vous n'utilisez pas cette commande), aucune méthode n'est utilisable dans l'éditeur de formules. Si une formule utilise un nom de méthode non autorisée, une erreur de syntaxe est générée et la formule ne peut pas être validée.

Passez dans le paramètre tabMéthodes le nom d'un tableau contenant la liste de méthodes à proposer dans l'éditeur de formules. Le tableau doit avoir été défini préalablement. Vous pouvez utiliser le caractère "joker" (@) dans les noms des méthodes afin de définir un ou plusieurs groupe(s) de méthodes autorisées.

Si vous souhaitez que l'utilisateur puisse appeler des commandes 4D non autorisées par défaut ou des commandes de plug-ins, vous devez utiliser des méthodes spécifiques chargées d'exécuter ces commandes.

Note : A compter de la version 2004.4 de 4D, le mécanisme de restriction d'accès aux commandes et méthodes dans l'éditeur de formules peut être désactivé pour tous les utilisateurs (option de compatibilité) ou pour le Super_Utilisateur et l'Administrateur via deux options des Préférences. Si l'option de compatibilité est cochée, la commande FIXER METHODES AUTORISEES est sans effet.

Exemple

Cet exemple autorise toutes les méthodes dont le nom débute par "formule" et de la méthode "Total_général" dans l'éditeur de formules :

```
TABLEAU ALPHA(15;tabméthodes;2)
tabméthodes{1}:="formule@"
tabméthodes{2}:="Total_général"
FIXER METHODES AUTORISEES(tabméthodes)
```

Référence

EDITER FORMULE, LIRE METHODES AUTORISEES.

LIRE METHODES AUTORISEES (tabMéthodes)

Paramètre	Type		Description
tabMéthodes	Tableau Alpha	←	Tableau de noms de méthodes

Description

La commande LIRE METHODES AUTORISEES retourne dans le tableau tabMéthodes le nom des méthodes “autorisées” dans l’éditeur de formules, c’est-à-dire pouvant être utilisées lors de l’écriture d’une formule — ces méthodes sont listées à la fin de la liste des commandes dans l’éditeur.

Par défaut, aucune méthode n’est utilisable dans l’éditeur de formules. Les méthodes doivent avoir été explicitement autorisées via la commande FIXER METHODES AUTORISEES. Si cette commande n’a pas été exécutée, LIRE METHODES AUTORISEES retourne une chaîne vide.

LIRE METHODES AUTORISEES retourne précisément ce qui a été passé à la commande FIXER METHODES AUTORISEES, c’est-à-dire un tableau alpha (la commande crée et dimensionne le tableau). En outre, si le caractère “joker” (@) a été utilisé pour désigner un groupe de méthodes, la chaîne contenant le caractère @ est retournée (et non les noms des méthodes du groupe).

Cette commande est utile pour préserver le paramétrage de l’ensemble courant de méthodes autorisées avant l’exécution d’une formule dans un contexte spécifique (par exemple un état rapide).

Exemple

Cet exemple permet d’autoriser ponctuellement un ensemble de méthodes spécifiques pour la création d’un état rapide :

```
    `Stockage du paramétrage courant
LIRE METHODES AUTORISEES(tabméthodes)

    `Définition des méthodes pour l'état
tabméthodes_Etats{1}:="Etats_@"
FIXER METHODES AUTORISEES(tabméthodes_Etats)
QR ETAT([Personnes];"MonEtat")
```

`Rétablissement des paramètres courants
FIXER METHODES AUTORISEES(tabméthodes)

Référence

FIXER METHODES AUTORISEES.

26

Gestion de la saisie

ALLER A CHAMP ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Champ | Variable	→	Nom d'objet (si * spécifié) sinon Variable ou champ saisissable à sélectionner

Description

La commande ALLER A CHAMP permet de sélectionner l'objet saisissable objet (variable ou champ) en tant que zone active du formulaire. C'est l'équivalent d'un clic de l'utilisateur dans la zone ou de l'utilisation de la touche **Tabulation** pour sélectionner le champ ou la variable.

Si vous passez le paramètre optionnel *, vous indiquez que le second paramètre désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables texte uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Pour supprimer tout focus dans le formulaire courant, appelez la commande en passant un nom d'objet vide dans objet (cf. exemple 2).

Note : Cette commande fonctionne uniquement dans un contexte de saisie dans un formulaire. Elle ne fait rien lorsqu'elle est utilisée avec des zones de saisie situées dans un sous-formulaire en mode liste.

Exemples

(1) Voici les deux modes d'utilisation de la commande ALLER A CHAMP :

```
ALLER A CHAMP ([Personnel]Nom)   `Référence de champ
ALLER A CHAMP (*;"ZonePrénoms")   `Nom d'objet
```

(2) Vous souhaitez que plus aucun objet du formulaire n'ait le focus :

```
ALLER A CHAMP (*;"")
```

(3) Reportez-vous à l'exemple de la commande REFUSER.

Référence
REFUSER.

EDITER ELEMENT ({*; }objet{; élément})

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un table ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * spécifié) ou Table ou variable (si * omis)
élément	Numérique	→	Numéro d'élément

Description

La commande EDITER ELEMENT permet de passer en "mode édition" l'élément courant ou l'élément de numéro élément du tableau ou de la liste désigné(e) par le paramètre objet. Le mode édition signifie que l'élément est sélectionné et prêt à être modifié : la saisie d'un caractère remplacera intégralement le contenu de l'élément.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (dans ce cas, passez une chaîne dans objet). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est une table ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de table ou une variable.

Cette commande s'applique aux objets saisissables suivants :
• Listes hiérarchiques
• List box
• Sous-formulaires (dans ce cas, seul un nom d'objet — le sous-formulaire — peut être passé dans objet),
• Formulaires liste affichés via la commande MODIFIER SELECTION ou VISUALISER SELECTION.

Si la commande est utilisée avec un objet saisissable qui n'est pas une liste, elle équivaut à la commande ALLER A CHAMP.
La commande ne fait rien si la liste ou le tableau désigné(e) est vide ou invisible. Si la liste ou le tableau n'est pas saisissable, la commande sélectionne (sans passer en édition) l'élément spécifié.
Dans le cadre d'une list box, si la colonne n'autorise pas la saisie de texte (saisie par case à cocher ou menu déroulant uniquement), l'élément spécifié prend le focus.

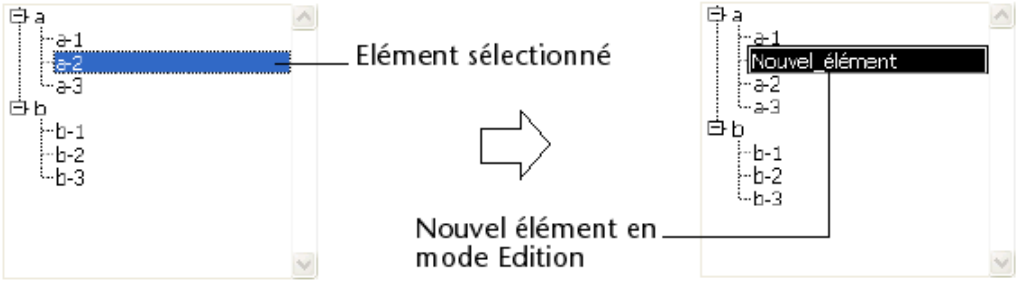
Le paramètre facultatif élément vous permet de désigner la position de l'élément (liste hiérarchique) ou le numéro de la ligne (list box, formulaire liste et sous-formulaire en mode "multi-sélection") à passer en édition. Si vous ne passez pas ce paramètre, la commande s'applique à l'élément courant de l'objet. S'il n'y a pas d'élément courant, le premier élément de l'objet passe en édition.

Note : Dans le cadre des sous-formulaires et des formulaires liste, la commande passe en édition le premier champ de la ligne spécifiée, dans l'ordre de saisie.

Exemples

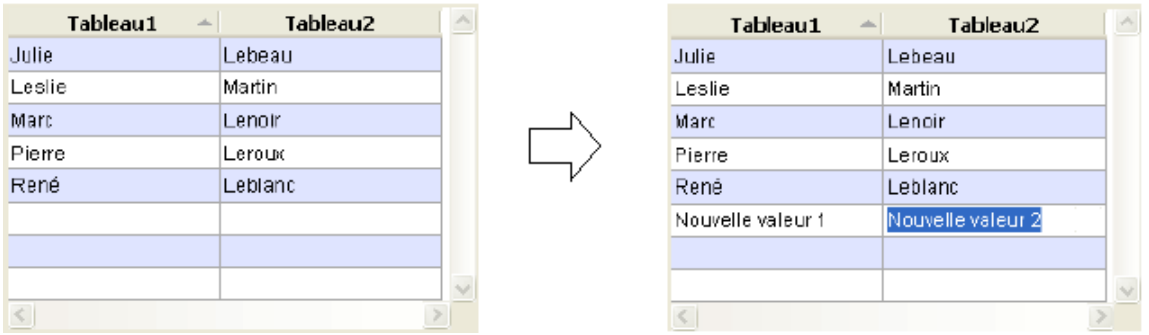
(1) Cette commande peut être utile notamment lors de la création d'un nouvel élément de liste hiérarchique. Au moment de l'appel de la commande, le dernier élément ajouté ou inséré dans la liste devient automatiquement éditable, sans que l'utilisateur n'ait à effectuer d'action spécifique.
Le code suivant pourrait être la méthode d'un bouton permettant d'insérer un nouvel élément dans une liste existante. Le libellé "Nouvel_élément" proposé par défaut est automatiquement placé en mode édition :

```
vlUniqueRef:=vlUniqueRef+1
INSERER DANS LISTE(hList;*;"Nouvel_élément";vlUniqueRef)
EDITER ELEMENT(*;"MaListe")
```



(2) Soient deux colonnes d'une list box dont les noms de variables associées sont respectivement "Tableau1" et "Tableau2". L'exemple suivant insère un nouvel élément dans les deux tableaux et passe le nouvel élément du tableau 2 en mode édition :

```
$vlNumLigne:=Taille tableau(Tableau1)+1
INSERER LIGNE LISTBOX(*;"MaListBox";$vlNumLigne)
Tableau1{$vlNumLigne}:="Nouvelle valeur 1"
Tableau2{$vlNumLigne}:="Nouvelle valeur 2"
EDITER ELEMENT(Tableau2; $vlNumLigne)
```



(3) L'exemple suivant permet de passer en mode édition le premier champ du dernier sous-enregistrement de la sous-sélection :

```
ALLER A DERNIER SOUS ENREGISTREMENT([Enfants])
EDITER ELEMENT(*;"Sousform")
```

Référence

ALLER A CHAMP, CHANGER ELEMENT, INSERER ELEMENT.

FILTRER FRAPPE CLAVIER (carFiltré)

Paramètre	Type		Description
carFiltré	Alpha	→	Caractère(s) de remplacement ou Chaîne vide pour annuler le filtrage clavier

Description

FILTRER FRAPPE CLAVIER vous permet de remplacer le caractère saisi par l'utilisateur dans un champ ou une zone saisissable par le premier caractère de la chaîne carFiltré.

Si vous passez une chaîne vide, le filtrage clavier en cours est annulé.

Vous appelez généralement FILTRER FRAPPE CLAVIER dans une méthode formulaire ou objet lorsque vous gérez l'événement formulaire Sur avant frappe clavier. Pour détecter les événements de frappe clavier, utilisez la commande Evenement formulaire. Pour récupérer les caractères saisis au clavier, utilisez les fonctions Frappe clavier ou Lire texte edite.

IMPORTANT : Si vous voulez effectuer des opérations “à la volée” en fonction de la valeur courante de la zone saisissable en cours de modification ainsi que du caractère à saisir, rappelez-vous que le texte affiché à l'écran n'est pas encore la valeur du champ ou de la variable. La valeur saisie dans une variable ou un champ ne lui est affectée que lorsque la zone est validée (lorsque l'utilisateur appuie sur la touche **Tabulation**, clique sur un bouton, etc.). En conséquence, pensez à placer les valeurs saisies dans une variable temporaire et à travailler avec celle-ci, puis à assigner cette variable à la zone de saisie (reportez-vous à l'exemple ci-dessous). Vous pouvez également utiliser la fonction Lire texte edite.

Utilisez la commande FILTRER FRAPPE CLAVIER dans les cas suivants :
• Pour effectuer un filtrage personnalisé des caractères,
• Pour créer un filtre de saisie non disponible en standard,
• Pour implémenter des zones de recherche ou de pré-saisie dynamiques.

ATTENTION : si vous appelez la commande Frappe clavier après avoir appelé FILTRER FRAPPE CLAVIER, c'est le caractère passé à cette commande qui sera retourné et non le caractère réellement saisi.

Exemples

(1) Avec le code suivant :

```
    ` Méthode objet de la zone saisissable monObjet
Au cas ou
    : (Evenement formulaire=Sur chargement )
    monObjet:=""
    : (Evenement formulaire=Sur avant frappe clavier )
        Si(Position(Frappe clavier;"0123456789")>0)
            FILTRER FRAPPE CLAVIER("*")
        Fin de si
Fin de cas
```

... tous les chiffres saisis dans la zone monObjet seront transformés en astérisques.

(2) Le code ci-dessous définit le comportement d'une zone de saisie de mot de passe, dans laquelle les caractères saisis sont remplacés à l'écran par des caractères aléatoires :

```
    ` Méthode objet de la zone saisissable vaMotsPasse
Au cas ou
    : (Evenement formulaire=Sur chargement)
        vaMotsPasse:=""
        vaMotPasseActuel:=""
    : (Evenement formulaire=Sur avant frappe clavier)
        Gérer frappe clavier (->vaMotsPasse;->vaMotPasseRéel)
        Si (Position(Frappe clavier;Caractere(Touche Retour arrière)+
                                    Caractere(Touche gauche )+Caractere(Touche droite )+
                                      Caractere(Touche haut )+Caractere(Touche bas ))=0)
            FILTRER FRAPPE CLAVIER(Caractere(65+(Hasard%26)))
        Fin de si
Fin de cas
```

Une fois la zone validée, vous récupérez le mot de passe réellement saisi par l'utilisateur dans la variable vaMotPasseRéel. La méthode Gérer frappe clavier est listée dans l'exemple de la commande Frappe clavier.

(3) Vous disposez dans votre application de diverses zones de texte dans lesquelles vous pouvez saisir quelques phrases. Votre application comporte également une table de glossaire contenant les termes les plus fréquemment utilisés dans votre base. Lors de l'édition de vos zones de texte, vous voulez pouvoir rapidement, à partir du glossaire, retrouver et insérer des mots en fonction des caractères sélectionnés dans le texte. Pour cela, vous avez deux solutions : soit placer des boutons avec des touches associées qui vont exécuter l'opération, soit intercepter les frappes clavier spéciales pendant la saisie. L'exemple ci-dessous utilise la seconde solution, basée sur la touche **Aide**.

Comme décrit ci-dessus, lorsque vous éditez une zone de texte, la valeur du champ ou de la variable de texte ne sera réellement modifiée que lorsque que vous l'aurez validée. Pour retrouver et insérer rapidement des entrées du glossaire dans une zone de texte alors qu'elle est en train d'être modifiée, vous devez donc créer une seconde zone "tampon" pour y placer les valeurs saisies. Vous pouvez effectuer cette opération à l'aide de la méthode projet décrite ci-dessous. Vous passez comme premiers paramètres des pointeurs vers la zone de saisie et vers la variable, puis la chaîne de caractère “interdits” comme troisième paramètre. Peu importe comment l'entrée clavier sera traitée, la méthode retourne la valeur saisie originale. Les caractères “interdits” sont les caractères que vous ne voulez pas insérer dans la zone saisissable et que vous voulez traiter en tant que caractères spéciaux.

```
    ` Méthode projet Frappe clavier tampon
    ` Frappe clavier tampon ( Pointeur ; Pointeur ; Alpha ) -> Alpha
    ` Frappe clavier tampon ( -> zoneSource ; -> valeurCourante ; Filtre )
    ` -> Ancien frappe clavier
C_ALPHA(1;$0)
C_POINTEUR($1;$2)
C_TEXTE($vtNouvValeur)
C_ALPHA(255;$3)
    ` Retourne la frappe clavier originale
$0:=Frappe clavier
    ` Obtenir la sélection de texte dans la zone saisissable
TEXTE SELECTIONNE($1->;$vlDébut;$vlFin)
    ` Commencer à travailler sur la valeur courante
$vtNouvValeur:=$2->
    ` En fonction de la touche enfoncée ou du caractère saisi,
    ` effectuer les actions appropriées
Au cas ou
        ` La touche Retour arrière a été enfoncée
    : (Code de caractere($0)=Touche Retour arrière )
```

```
                ` Supprimer les caractères sélectionnés ou le caractère à gauche du curseur
            $vtNouvValeur:=Supprimer texte ($vtNouvValeur;$vlDébut;$vlFin)
                ` Une touche "flèche" a été appuyée
                ` Ne faites rien sauf accepter la frappe clavier
        : (Code de caractere($0)=Touche gauche )
        : (Code de caractere($0)=Touche droite )
        : (Code de caractere($0)=Touche haut )
        : (Code de caractere($0)=Touche bas )

            ` Un caractère valide a été saisi
        : (Position($0;$3)=0)
            $vtNouvValeur:=Inserer texte ($vtNouvValeur;$vlDébut;$vlFin;$0)
    Sinon
            ` Le caractère n'est pas accepté
        FILTRER FRAPPE CLAVIER("")
    Fin de cas
        ` Retourner la valeur pour la prochaine gestion de la frappe clavier
    $2->:=$vtNouvValeur
```

Cette méthode utilise les sous-méthodes suivantes :

```
        ` Méthode projet Supprimer texte
        ` Supprimer texte ( Alpha ; Long ; Long ) -> Alpha
        ` Supprimer texte ( -> Texte ; SelDébut ; SelFin ) -> Nouveau texte
    C_TEXTE($0;$1)
    C_ENTIER LONG($2;$3)
    $0:=Sous chaine($1;1;$2-1-Num($2=$3))+Sous chaine($1;$3)

        ` Méthode projet Inserer texte
        ` Inserer texte ( Alpha ; Long ; Long ; Alpha ) -> Alpha
        ` Inserer texte ( -> texteSource ; SelDébut ; SelFin ; Texte à insérer ) -> Nouveau texte
    C_TEXTE($0;$1;$4)
    C_ENTIER LONG($2;$3)
    $0:=$1
    Si ($2#$3)
        $0:=Sous chaine($0;1;$2-1)+$4+Sous chaine($0;$3)
    Sinon
        Au cas ou
            : ($2<=1)
                $0:=$4+$0
```

```
        : ($2>Longueur($0))
            $0:=$0+$4
        Sinon
            $0:=Sous chaine($0;1;$2-1)+$4+Sous chaine($0;$2)
        Fin de cas
    Fin de si
```

Une fois que vous avez ajouté ces méthodes projet à votre base, vous pouvez les utiliser de la manière suivante :

```
        ` Méthode objet de la zone saisissable vaDescription
    Au cas ou
        : (Evenement formulaire=Sur chargement )
            vaDescription:=""
            vaDescriptionDouble:=""
                ` Etablir la liste des caractères "interdits" à traiter comme des touches spéciales
                ` (Dans cet exemple, seule la touche Aide est filtrée)
            vaTouchesSpéciales:=Caractere(Touche Aide)
        : (Evenement formulaire=Sur avant frappe clavier)
            $vsKey:=Frappe clavier tampon (->vaDescription;->vaDescriptionDouble;
                                                                vaTouchesSpéciales)
            Au cas ou
                : (Code de caractere($vsKey)=Touche Aide)
                        ` Faire quelque chose lorsque la touche Aide est enfoncée
                        ` Dans cet exemple, une saisie de glossaire doit être recherchée
                        ` et insérée
                    chercher_Glossaire (->vaDescription;->vaDescriptionDouble)
            Fin de cas
    Fin de cas
```

La méthode projet chercher_Glossaire est listée ci-dessous (le point principal est l'utilisation de la variable tampon pour réaffecter la zone saisissable à modifier) :

```
    ` Méthode projet chercher_Glossaire
    ` chercher_Glossaire ( Pointeur ; Pointeur )
    ` chercher_Glossaire ( -> zone saisissable ; ->variable double )

C_POINTEUR($1;$2)
C_ENTIER LONG($vlDébut;$vlFin)
```

```
    ` Obtenir la sélection de texte dans la zone saisissable
TEXTE SELECTIONNE($1->;$vlDébut;$vlFin)
    ` Obtenir le texte sélectionné ou le mot situé à gauche du curseur
$vtTexteSelectionne:=obtenirTexteSelectionne ($2->;$vlDébut;$vlFin)
    ` Y a-t-il quelque chose à rechercher ?
Si ($vtTexteSelectionne#"")
        ` Si la sélection de texte était le curseur, la sélection débute au mot situé
        ` après le curseur
    Si ($vlDébut=$vlFin)
        $vlDébut:=$vlDébut-Longueur($vtTexteSelectionne)
    Fin de si
        ` Chercher la première entrée du glossaire disponible
    CHERCHER([Glossaire];[Glossaire]Saisie=$vtTexteSelectionne+"@")
        ` Existe-t-elle ?
    Si (Enregistrements trouves([Glossaire])>0)
            ` Si oui, l'insérer dans la zone tampon
        $2->:=Inserer texte ($2->;$vlDébut;$vlFin;[Glossaire]Saisie)
            ` Copier le tampon dans la zone saisissable
        $1->:=$2->
            ` Fixer la sélection après avoir inséré l'entrée du glossaire
        $vlFin:=$vlDébut+Longueur([Dictionnaire]Saisie)
        SELECTIONNER TEXTE(vsComments;$vlFin;$vlFin)
    Sinon
            ` Il n'y a pas d'entrée qui correspond dans le glossaire
        BEEP
    Fin de si
Sinon
        ` Il n'y a pas de texte sélectionné
    BEEP
Fin de si
```

La méthode obtenirTexteSelectionne est la suivante :

```
    ` Méthode objet obtenirTexteSelectionne
    ` obtenirTexteSelectionne ( Alpha ; Entier long ; Entier long ) -> Alpha
    ` obtenirTexteSelectionne ( Texte ; SelDébut ; SelFin ) -> texte sélectionné
C_TEXTE($0;$1)
C_ENTIER LONG($2;$3)
Si ($2<$3)
    $0:=Sous chaine($1;$2;$3-$2)
Sinon
```

```
            $0:=""
            $2:=$2-1
            Repeter
                Si ($2>0)
                    Si (Position($1[[$2]];"  ,.!?:;()-_——")=0)
                        $0:=$1[[$2]]+$0
                        $2:=$2-1
                    Sinon
                        $2:=0
                    Fin de si
                Fin de si
            Jusque ($2=0)
        Fin de si
```

Référence

Evenement formulaire, Frappe clavier, Lire texte edite.

Frappe clavier → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Caractère saisi par l'utilisateur

Description

Frappe clavier retourne le caractère tapé par l'utilisateur dans un champ ou une zone saisissable.

En général, vous appelez Frappe clavier dans une méthode formulaire ou objet, lors de la gestion des événements formulaire Sur avant frappe clavier et Sur après frappe clavier. Pour détecter les événements de frappe clavier, utilisez la commande Evenement formulaire.

Si vous voulez remplacer un caractère saisi par l'utilisateur par un autre, utilisez la commande FILTRER FRAPPE CLAVIER.

IMPORTANT : Si vous voulez effectuer des opérations “à la volée” en fonction de la valeur courante de la zone saisissable en cours de modification ainsi que du caractère à saisir, rappelez-vous que le texte affiché à l'écran n'est pas encore la valeur du champ ou de la variable. La valeur saisie dans une variable ou un champ ne lui est affectée que lorsque la zone est validée (si l'utilisateur appuie sur la touche Tabulation, clique sur un bouton, etc.). En conséquence, pensez à placer les valeurs saisies dans une variable temporaire et à travailler avec celle-ci, ou utilisez la commande Lire texte edite. Vous devez procéder ainsi si vous souhaitez connaître la valeur courante du texte pour effectuer des actions spéciales.

Vous pouvez utiliser la commande Frappe clavier pour :
• effectuer un filtrage personnalisé des caractères
• créer un filtre de saisie non disponible en standard, par exemple dans les filtres de saisie
• implémenter des zones de recherche ou de pré-saisie dynamiques.

Note : Vous ne pouvez pas utiliser la fonction Frappe clavier dans les sous-formulaires.

Exemples

(1) Référez-vous aux exemples de la commande FILTRER FRAPPE CLAVIER.

(2) Lorsque vous traitez un événement Sur avant frappe clavier, vous gérez la modification de la zone de texte courante (celle qui contient le curseur), et non la “valeur future” de la source de données (champ ou variable) de cette zone. La méthode Gérer frappe clavier décrite ci-dessous vous permet de placer dans une seconde variable les caractères saisis dans une zone de texte. Vous pouvez alors utiliser cette variable pour effectuer différentes actions pendant la saisie des caractères dans la zone. Vous passez comme premier paramètre un pointeur vers la source des données de la zone, et comme second paramètre un pointeur vers cette seconde variable. La méthode renvoie la nouvelle valeur de la zone de texte dans la seconde variable et retourne Vrai si cette valeur est différente de ce qu'elle était avant la saisie du dernier caractère.

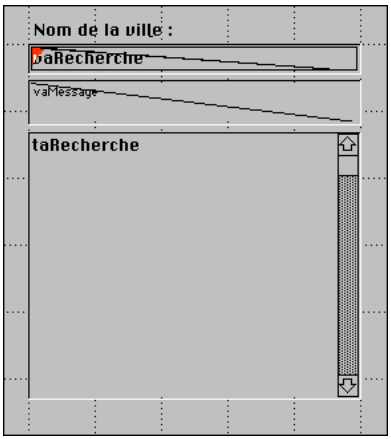
```
` Méthode projet Gérer frappe clavier
` Gérer frappe clavier ( Pointeur ; Pointeur ) -> Booléen
` Gérer frappe clavier ( -> zoneSource ; -> valeurCourante ) -> Nouvelle valeur

C_POINTEUR ($1;$2)
C_TEXTE ($vtNouvValeur)

   ` Récupérer le texte sélectionné dans la zone saisissable
TEXTE SELECTIONNE ($1->;$vlDébut;$vlFin)
   ` Commencer à travailler avec la valeur courante
$vtNouvValeur:=$2->
   ` Selon la touche appuyée ou le caractère saisi, effectuer les actions appropriées
Au cas ou

      ` La touche Retour arrière a été enfoncée
   : (Code de caractere(Frappe clavier)=Touche Retour arrière )
         ` Supprimer les caractères sélectionnés ou le caractère à gauche du curseur
      $vtNouvValeur:=Sous chaine ($vtNouvValeur;1;$vlDébut-1-Num($vlDébut=
                                          $vlFin))+Sous chaine($vtNouvValeur;$vlFin)

      ` Un caractère acceptable a été saisi
   : (Position (Frappe clavier;"abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
      Si ($vlDébut#$vlFin)
            ` Un ou plusieurs caractères sont sélectionnés, la frappe clavier va les
            ` effacer
         $vtNouvValeur:=Sous chaine($vtNouvValeur;1;$vlDébut-1) +
                                    Frappe clavier+Sous chaine($vtNouvValeur;$vlFin)
      Sinon
            ` La sélection de texte est le curseur
         Au cas ou
               ` Le curseur est actuellement au début du texte
            : ($vlDébut<=1)
                  ` Insertion du caractère au début du texte
               $vtNouvValeur:=Frappe clavier+$vtNouvValeur
                                                   ` Le curseur est
```

```
                : ($vlDébut>=Longueur($vtNouvValeur))
                        ` Ajouter le caractère à la fin du texte
                    $vtNouvValeur:=$vtNouvValeur+Frappe clavier
            Sinon
                    ` Le curseur se trouve dans le texte, insérer le nouveau caractère
                $vtNouvValeur:=Sous chaine($vtNouvValeur;1;$vlDébut-1)+
                                Frappe clavier+Sous chaine($vtNouvValeur;$vlDébut)
            Fin de cas
        Fin de si

        ` Une touche flèche a été enfoncée
        ` Ne rien faire, mais valider la frappe clavier
    : (Code de caractere(Frappe clavier)=Touche gauche )
    : (Code de caractere(Frappe clavier)=Touche droite )
    : (Code de caractere(Frappe clavier)=Touche haut )
    : (Code de caractere(Frappe clavier)=Touche bas )
        `
Sinon
        ` Ne pas accepter des caractères autres que des lettres, chiffres, espaces et tirets
    FILTRER FRAPPE CLAVIER ("")
Fin de cas
    ` Est-ce que la valeur est maintenant différente ?
$0:=($vtNouvValeur#$2->)
    ` Retourner la valeur pour la gestion de la prochaine frappe clavier
$2->:=$vtNouvValeur
```

Une fois que vous avez ajouté cette méthode projet à votre application, vous pouvez l'utiliser ainsi :

```
    ` Méthode objet de la zone saisissable MonObjet
Au cas ou
    : (Evenement formulaire=Sur chargement)
        MonObjet:=""
        MonObjetCaché:=""
    : (Evenement formulaire=Sur avant frappe clavier)
        Si (Gérer frappe clavier (->MonObjet;->MonObjetCaché))
            ` Effectuer des actions appropriées par rapport à la valeur stockée dans
            ` MonObjetCaché
        Fin de si
Fin de cas
```

Examinons par exemple le formulaire suivant :

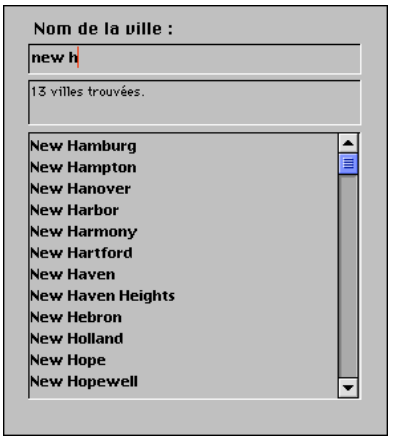


Il est composé des objets suivants : une zone saisissable vaRecherche, une zone non-saisissable vaMessage et une zone de défilement taRecherche. Lorsque l'utilisateur saisit des caractères dans vaRecherche, la méthode objet effectue une recherche sur la table [Codes postaux] permettant d'afficher des villes américaines en saisissant seulement les premiers caractères de leur nom. Voici la méthode objet de vaRecherche :

```
      ` Méthode objet de la zone saisissable vaRecherche
Au cas ou
   : (Evenement formulaire=Sur chargement )
      vaRecherche:=""
      vaRésultat:=""
      vaMessage:="Saisissez les premiers caractères de la ville que vous cherchez."
      EFFACER VARIABLE(taRecherche)
   : (Evenement formulaire=Sur avant frappe clavier )
      Si (Gérer frappe clavier (->vaRecherche;->vaRésultat))
         Si (vaRésultat#"")
            CHERCHER([Codes postaux];[Codes postaux]Ville=vaRésultat+"@")
            SUPPRIMER MESSAGES
            VALEURS DISTINCTES([Codes postaux]Ville;taRecherche)
            LAISSER MESSAGES
            $vlRésultat:=Taille tableau(taRecherche)
            Au cas ou
               : ($vlRésultat=0)
                  vaMessage:="Aucune ville trouvée."
               : ($vlRésultat=1)
                  vaMessage:="Une ville trouvée."
            Sinon
               vaMessage:=Chaine($vlRésultat)+" villes trouvées."
                                                Fin de cas
```

```
            Sinon
                SUPPRIMER DANS TABLEAU(taRecherche;1;
                                                Taille tableau(taRecherche))
                vaMessage:="Saisissez les premières lettres de la ville que vous cherchez."
            Fin de si
        Fin de si
    Fin de cas
```

Voici le formulaire en exécution :



A l'aide des possibilités de communication interprocess de 4D, vous pouvez construire une interface dans laquelle les recherches se construisent dans des palettes flottantes communiquant avec les process dans lesquels les enregistrements sont affichés ou modifiés.

Référence

Evenement formulaire, FILTRER FRAPPE CLAVIER, Lire texte edite.

NE PAS VALIDER

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande NE PAS VALIDER doit être employée dans une méthode objet ou formulaire (ou une sous-routine) pour :

• annuler la création ou la modification d'un enregistrement ou un sous-enregistrement — dont les données ont été saisies à la suite d'un AJOUTER ENREGISTREMENT, MODIFIER ENREGISTREMENT, AJOUTER SOUS ENREGISTREMENT ou MODIFIER SOUS ENREGISTREMENT.
• annuler un formulaire affiché par l'intermédiaire de la commande DIALOGUE.
• quitter un formulaire affichant une sélection d'enregistrements — à l'aide de VISUALISER SELECTION ou MODIFIER SELECTION.
• annuler l'impression d'une ligne sur le point d'être imprimée à l'aide de la commande Imprimer ligne (voir ci-dessous).

Dans le contexte de la saisie, NE PAS VALIDER effectue la même action que lorsque l'utilisateur utilise la touche d'annulation (**Esc**).
NE PAS VALIDER est fréquemment exécutée à la suite de la sélection d'une commande de menu. NE PAS VALIDER est également souvent appelée dans la méthode objet d'un bouton auquel la propriété "Pas d'action" a été associée.

Cette commande peut également être placée dans la méthode de la case de fermeture (optionnelle) d'une fenêtre créée par la commande Creer fenetre. Si la fenêtre comporte une case de menu Système, NE PAS VALIDER et VALIDER peuvent être appelées dans la méthode à exécuter lorsque l'utilisateur double-clique sur la case du menu Système ou sélectionne la commande de menu **Fermeture**.
Il n'est pas possible d'enchaîner plusieurs NE PAS VALIDER. En d'autres termes, l'exécution consécutive de deux commandes NE PAS VALIDER dans une méthode en réponse à un événement aura le même résultat que l'exécution d'une seule.

Enfin, cette commande peut être utilisée dans l'événement formulaire Sur impression corps, dans le cadre de l'utilisation de la commande Imprimer ligne. Dans ce contexte, la commande NE PAS VALIDER suspend l'impression de la ligne sur le point d'être imprimée, puis la reprend page suivante. Ce mécanisme permet de gérer le manque de place ou les sauts de page lors des impressions des lignes.

Note : Ce fonctionnement est différent de celui de l'instruction SAUT DE PAGE(*) qui provoque l'annulation de TOUTES les lignes en attente d'impression.

Exemple

Reportez-vous à l'exemple de la commande FIXER TAQUET IMPRESSION.

Référence

Imprimer ligne, SAUT DE PAGE, VALIDER.

Variables et ensembles système

Lorsque la commande NE PAS VALIDER est exécutée (formulaire annulé ou annulation d'impression), la variable système OK prend la valeur 0.

REFUSER {(champ)}

Paramètre	Type		Description
champ	Champ	→	Champ dont la saisie doit être refusée

Description

REFUSER accepte deux syntaxes. Dans la première syntaxe, REFUSER n'a pas de paramètre. Dans ce cas, la commande rejette la totalité de la saisie et force l'utilisateur à rester dans le formulaire. La seconde syntaxe permet de ne refuser que champ et force l'utilisateur à rester dans le champ.

Note : Nous vous conseillons d'utiliser en priorité les outils intégrés de validation de saisie de 4D, avant de faire appel à cette commande.

La première syntaxe de REFUSER est utilisée pour empêcher l'utilisateur de valider un enregistrement incomplet. Vous pouvez parvenir au même résultat sans utiliser REFUSER : associez la touche **Entrée** à un bouton n'effectuant "Pas d'action" et utilisez les commandes VALIDER et NE PAS VALIDER pour valider ou annuler l'enregistrement, une fois que les champs ont été correctement remplis. Il est recommandé d'employer cette seconde technique plutôt que d'utiliser la première syntaxe de REFUSER.

En général, vous employez la première syntaxe de REFUSER pour empêcher l'utilisateur de valider un enregistrement incomplet ou comportant des valeurs incorrectes. Si l'utilisateur tente de valider l'enregistrement, l'exécution de REFUSER provoque l'annulation de cette commande et l'enregistrement reste affiché dans le formulaire. L'utilisateur doit alors recommencer la saisie jusqu'à ce que les valeurs soient considérées comme correctes ou annuler l'enregistrement.

Le meilleur emplacement pour la commande REFUSER, lorsque vous utilisez cette syntaxe, est la méthode objet d'un bouton de type Valider associé à la touche de validation. De cette manière, la validation n'est possible que lorsque l'enregistrement est accepté, et l'utilisateur ne peut pas "forcer" la validation en appuyant sur la touche **Entrée**.

La seconde syntaxe de REFUSER utilise le paramètre champ. Si elle est exécutée, le curseur reste dans la zone de saisie du champ. Cette syntaxe oblige l'utilisateur à saisir une valeur correcte. Cette instruction doit être appelée juste après la modification du champ.

Vous pouvez tester la modification d'un champ à l'aide de la fonction Modifie. Vous pouvez également placer la commande REFUSER dans la méthode objet de la zone de saisie. Lorsqu'elle est utilisée avec des champs de sous-formulaires, cette commande ne fait rien.

Vous devez placer cette syntaxe de REFUSER soit dans la méthode formulaire, soit dans une méthode objet du formulaire en train d'être modifié. Si vous utilisez REFUSER avec le formulaire "pleine page" d'un sous-formulaire, placez-la dans la méthode formulaire ou une méthode objet du formulaire "pleine page".

Vous pouvez utiliser la commande SELECTIONNER TEXTE pour sélectionner, à l'intérieur du champ, les valeurs qui ont été refusées.

Exemples

(1) L'exemple suivant illustre la première syntaxe de REFUSER, placée dans la méthode objet d'un bouton Valider. La touche **Entrée** a été définie comme équivalent clavier pour ce bouton. Cela signifie que même si l'utilisateur appuie sur cette touche pour valider l'enregistrement, la méthode objet du bouton sera exécutée. L'enregistrement est une transaction bancaire. Si la transaction est un chèque, un numéro de chèque doit être saisi. S'il n'y a pas de numéro, la validation est refusée :

```
Au cas ou
: (([Opération]Trans = "Chèque") & ([Opération]Numéro = ""))
        ` Si c'est un chèque sans numéro...
    ALERTE ("Veuillez saisir le numéro du chèque.")  ` Alerter l'utilisateur
    REFUSER  ` Refuser la saisie
    ALLER A CHAMP ([Opération]Numéro)
        ` Placer le curseur dans le champ "numéro de chèque"
Fin de cas
```

(2) L'exemple suivant est une partie de la méthode objet d'un champ [Employés]Salaire. La méthode objet teste si la valeur de ce champ est inférieure à 10 000 Euros et la refuse si c'est le cas. Vous pourriez effectuer le même contrôle en spécifiant une valeur minimum pour le champ, dans l'éditeur de formulaires du mode Développement :

```
Si ([Employés]Salaire<10000)
    ALERTE ("Le salaire annuel doit être supérieur à 10 000 Euros.")
    REFUSER ([Employés]Salaire)
Fin de si
```

Référence

ALLER A CHAMP, NE PAS VALIDER, VALIDER.

VALIDER

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande VALIDER doit être employée dans une méthode objet ou formulaire (ou une sous-routine) pour :

• valider un enregistrement ou un sous-enregistrement créé ou modifié — dont les données ont été saisies à la suite d'un AJOUTER ENREGISTREMENT, MODIFIER ENREGISTREMENT, AJOUTER SOUS ENREGISTREMENT ou MODIFIER SOUS ENREGISTREMENT.
• valider un formulaire affiché par l'intermédiaire de la commande DIALOGUE.
• quitter un formulaire affichant une sélection d'enregistrements — à l'aide de VISUALISER SELECTION ou MODIFIER SELECTION.

VALIDER effectue la même action que lorsque l'utilisateur appuie sur la touche **Entrée**. Une fois que le formulaire a été validé, la variable système OK prend la valeur 1.

VALIDER est fréquemment exécutée à la suite de la sélection d'une commande de menu. VALIDER est également souvent appelée dans la méthode objet d'un bouton auquel la propriété "Pas d'action" a été associée.

Enfin, cette commande peut être placée dans la méthode de la case de fermeture (optionnelle) d'une fenêtre créée par la commande Creer fenetre. Si la fenêtre comporte une case de menu Système, VALIDER et NE PAS VALIDER peuvent être appelées dans la méthode à exécuter lorsque l'utilisateur double-clique sur la case du menu Système ou sélectionne la commande de menu **Fermeture**.

Il n'est pas possible d'enchaîner plusieurs VALIDER. En d'autres termes, l'exécution consécutive de deux commandes VALIDER dans une méthode en réponse à un événement aura le même résultat que l'exécution d'une seule.

Référence

NE PAS VALIDER.

27

Glisser-Déposer

Présentation du Glisser-Déposer

4D dispose de fonctions intégrées vous permettant de gérer le glisser-déposer ("drag and drop") parmi les objets de vos formulaires et vos applications. Vous pouvez glisser-déposer un objet sur un autre objet situé dans la même fenêtre ou dans une autre fenêtre. Autrement dit, vous pouvez effectuer des glisser-déposer à l'intérieur d'un même process ou entre différents process.

Vous pouvez également glisser-déposer des objets entre les formulaires 4D et d'autres applications ou le bureau du système d'exploitation, ou effectuer l'opération inverse. Par exemple, il est possible de glisser-déposer un fichier image GIF dans un champ image 4D. Il est également possible de sélectionner du texte dans une application de traitement de texte et de le déposer dans une variable texte de 4D.

Enfin, il est possible de déposer des objets directement sur l'application sans qu'un formulaire ne soit nécessairement au premier plan. La Méthode base Sur déposer permet dans ce cas de gérer le glisser-déposer. Ce principe permet par exemple d'ouvrir un document 4D Write en le déposant sur l'icône de l'application 4D.

Note : Pour ne pas alourdir cette introduction, nous admettrons ici que le glisser-déposer permet de "transporter" des données d'un point à un autre. Nous verrons plus loin qu'un glisser-déposer peut aussi être la métaphore (c'est-à-dire la représentation au niveau de l'interface utilisateur) de toute opération, quelle qu'elle soit.

Propriétés d'objets "Glissable" et "Déposable"

Si vous souhaitez qu'un objet soit **glissable**, c'est-à-dire que vous puissiez le faire glisser et le déposer sur un autre objet, vous devez sélectionner la propriété "Glissable" pour cet objet dans la Liste des propriétés. L'objet que vous faites glisser est appelé **objet source** de l'opération de glisser-déposer.

Si vous souhaitez qu'un objet soit **déposable**, c'est-à-dire que l'objet puisse être la destination d'une opération de glisser-déposer, vous devez sélectionner la propriété "Déposable" pour cet objet dans la Liste des propriétés. L'objet qui reçoit les données est appelé **objet de destination** de l'opération de glisser-déposer.

Glisser automatique et **Déposer automatique** : ces propriétés supplémentaires sont disponibles pour les champs et variables texte, combo box et list box. L'option **Déposer automatique** est également disponible pour les champs et variables image. Elles permettent d'activer un mode de glisser-déposer automatique basé sur la copie du contenu (le glisser-déposer n'est plus géré par les événements formulaires 4D). Reportez-vous au pararaphe "Glisser-Déposer automatique" à la fin de ce chapitre.

Par défaut, les objets nouvellement créés ne possèdent aucune de ces propriétés. Il est de votre ressort de les sélectionner explicitement.

Tous les objets situés dans un formulaire entrée ou dans une boîte de dialogue peuvent être définis comme glissables et déposables. Les éléments individuels d'un tableau (par exemple une zone de défilement), les éléments d'une liste hiérarchique ou les lignes d'une list box peuvent être glissé(e)s et déposé(e)s. Inversement, vous pouvez faire glisser et déposer tout objet sur un élément individuel d'un tableau ou d'une liste hiérarchique, ou encore une ligne de list box. Il n'est toutefois pas possible de faire glisser et de déposer des objets depuis la zone de corps d'un formulaire sortie.

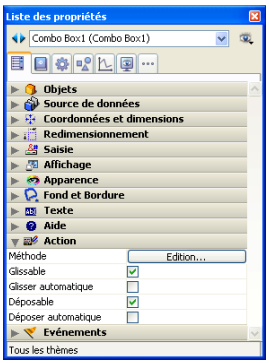
Vous pouvez également gérer les glisser-déposer sur l'application, en-dehors de tout formulaire, via la Méthode base Sur déposer.

Afin de vous laisser "carte blanche" lors de la construction d'une interface utilisateur exploitant le glisser-déposer, 4D vous permet d'utiliser tout type d'objet actif (champ ou variable) en tant qu'objet source ou destination. Par exemple, si vous le souhaitez, vous pouvez glisser-déposer des boutons.

Notes :
• Pour faire glisser un texte ou un bouton ayant la propriété "glissable", vous devez au préalable appuyer sur la touche **Alt** (Windows) ou **Option** (Mac OS).
• Par défaut, dans le cas des variables et champs image, l'image et sa référence sont glissées. Pour ne faire glisser que la référence de la variable ou du champ, appuyez au préalable sur la touche **Alt** (Windows) ou **Option** (Mac OS).
• Lorsque, pour un objet de type List box, les propriétés “Glissable” et “Ligne déplaçable” sont définies simultanément, la propriété “Ligne déplaçable” est prioritaire en cas de déplacement d’une ligne. Le glisser n’est pas possible dans ce cas.

Notez qu'un objet "glissable" et "déposable" peut être déposé sur lui-même (à moins que vous n'interdisiez cette opération, reportez-vous aux paragraphes suivants).

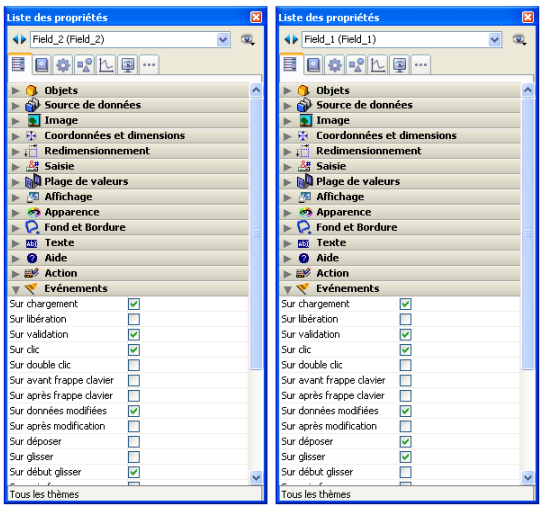
Voici la Liste des propriétés, dans laquelle les propriétés "Glissable" et "Déposable" ont été définies pour l'objet sélectionné :



Gérer le glisser-déposer par programmation

La gestion du glisser-déposer par programmation est basée sur trois événements formulaires : Sur début glisser, Sur glisser et Sur déposer.
A noter que l'événement Sur début glisser **est généré dans le contexte de l'objet source** du glisser tandis que Sur glisser et Sur déposer **sont envoyés à l'objet de destination uniquement**.

Pour que l'application puisse traiter ces événements, il doivent avoir été sélectionnés de manière appropriée dans la Liste des propriétés :



Sur début glisser

L'événement formulaire Sur début glisser est sélectionnable pour tous les objets de formulaire pouvant être glissés. Il est généré dans tous les cas lorsque l'objet dispose de la propriété **Glissable**.
A la différence de l'événement formulaire Sur glisser, Sur début glisser est appelé dans le contexte de l'objet à la source du glisser. Il peut être appelé dans la méthode de l'objet source ou la méthode du formulaire de l'objet source.
Cet événement est utile pour la gestion avancée du glisser. Il permet notamment de :
• lire les données et les signatures présentes dans le conteneur (via la commande LIRE DONNEES CONTENEUR).
• ajouter des données et des signatures dans le conteneur (via la commande AJOUTER DONNEES AU CONTENEUR).
• accepter ou refuser le glisser via $0 dans la méthode de l'objet glissé. Pour signaler que le glisser est accepté, la méthode de l'objet source doit retourner 0 (zéro), vous exécutez donc $0:=0. Pour signaler que le glisser est refusé, la méthode de l'objet source doit retourner -1 (moins un), vous exécutez donc $0:=-1. Si aucun résultat n'est retourné, 4D considère que le glisser est accepté.

Les données de 4D sont placées dans le conteneur de données avant l'appel de l'événement. Par exemple, pour un glisser sans l'option **Glisser automatique**, le texte glissé est déjà dans le conteneur au moment de l'appel de l'événement.

Sur glisser

L'événement Sur glisser est envoyé de manière répétée à l'objet de destination lorsque le pointeur de la souris est placé sur l'objet. Généralement, en réponse à cet événement, vous effectuez les actions suivantes :
• Vous appelez la commande PROPRIETES GLISSER DEPOSER qui vous renseigne sur l'objet source.
• En fonction de la nature et du type de l'objet de destination (celui duquel la méthode est en cours d'exécution) et de l'objet source, vous **acceptez** ou **refusez** le glisser.

Pour signaler que le glisser est accepté, la méthode de l'objet de destination doit retourner 0 (zéro), vous exécutez donc $0:=0. Pour signaler que le glisser est refusé, la méthode de l'objet de destination doit retourner -1 (moins un), vous exécutez donc $0:=-1. Pendant un événement Sur glisser, 4D traite la méthode de l'objet comme une fonction. Si aucun résultat n'est retourné, 4D considère que le glisser est accepté.

Si vous acceptez le glisser, l'objet de destination est activé. Si vous le refusez, l'objet de destination reste inactivé. Accepter le glisser ne signifie pas que les données glissées vont être insérées dans l'objet de destination. Cela signifie uniquement que l'objet de destination, si le bouton de la souris était relâché à cet instant, accepterait les données.

Si vous ne gérez pas l'événement Sur glisser pour un objet dont la propriété "Déposable" a été sélectionnée, l'objet sera activé pour tous les glisser, quels que soient la nature et le type des données glissées.

La traitement de l'événement Sur glisser vous permet de contrôler la première phase d'une opération de glisser-déposer : non seulement vous pouvez tester si le type des données glissées est compatible avec l'objet de destination — et donc accepter ou refuser le glisser — mais également, vous en informez l'utilisateur, car 4D active ou non l'objet de destination en fonction de votre décision.

Le code traitant un événement Sur glisser doit être court et s'exécuter rapidement car cet événement est envoyé de manière répétée à l'objet de destination courant, en fonction des mouvements de la souris.

ATTENTION : Si le glisser-déposer est un **glisser-déposer interprocess**, ce qui signifie que l'objet source est situé dans un process (fenêtre) différent de celui de l'objet de destination, la méthode de l'objet de destination lors de l'événement Sur glisser est exécutée **dans le contexte du process source** (le process de l'objet source) et non dans le process de l'objet de destination. C'est le seul cas où un tel type d'exécution a lieu (les avantages que procure ce fonctionnement sont décrits à la fin de cette section).

Sur déposer

L'événement Sur déposer est envoyé (une seule fois) à l'objet de destination lorsque le bouton de la souris est relâché alors que le pointeur se trouvait au-dessus de l'objet. Cet événement est la seconde phase d'un glisser-déposer, dans laquelle vous effectuez les véritables opérations répondant à l'action de l'utilisateur.

Cet événement n'est pas envoyé à l'objet si le glisser n'a pas été accepté dans le ou les événement(s) Sur glisser. Si vous traitez l'événement Sur glisser pour un objet et refusez le glisser, l'événement Sur déposer ne se déclenche pas. Ainsi, si pendant l'événement Sur glisser vous avez testé la compatibilité entre le type des données de l'objet source et de destination et accepté le glisser, vous n'avez pas besoin de tester de nouveau les données dans l'événement Sur déposer : vous savez déjà qu'elles sont compatibles.

L'aspect le plus intéressant de l'implémentation du glisser-déposer dans 4D est que le programme vous permet de faire ce que vous voulez. Par exemple :

• Si un élément de liste hiérarchique est déposé sur un champ de type Texte, vous pouvez décider d'insérer le texte de l'élément de liste au début, au milieu ou à la fin du champ de texte.

• Votre formulaire contient un bouton image à deux états représentant une corbeille vide et une corbeille pleine. Le glisser-déposer d'un objet sur ce bouton pourrait provoquer, du point de vue de l'interface utilisateur : “supprimer l'objet qui a été glissé-déposé dans la corbeille”. Ici, le glisser-déposer ne transporte pas des données d'un point à un autre, mais plutôt il déclenche une action.
• Glisser un élément de tableau depuis une palette flottante vers un objet dans un formulaire pourrait signifier “afficher dans cette fenêtre l'enregistrement du client dont le nom a été glissé-déposé depuis la fenêtre flottante listant les noms de tous les clients stockés dans la base”.
• Etc.

La gestion du glisser-déposer de 4D est une boîte à outils vous permettant d'implémenter toutes les métaphores d'interface utilisateur auxquelles vous pouvez penser.

Les commandes du thème Glisser-Déposer

La commande PROPRIETES GLISSER DEPOSER retourne :
• un pointeur vers l'objet glissé (champ ou variable),
• le numéro de l'élément de tableau ou de liste hiérarchique le cas échéant,
• le numéro du process source.

La commande Position deposer retourne le numéro ou la position de l'élément cible si l'objet de destination est un tableau (c'est-à-dire une zone de défilement), une liste hiérarchique, un texte ou une combo box, ainsi que le numéro de colonne si l'objet est une list box.

Les commandes telles que RESOUDRE POINTEUR et Type sont utiles pour tester la nature et le type de l'objet source.

Lorsque l'opération de glisser-déposer est destinée à copier les données glissées :
• Si le glisser-déposer est effectué à l'intérieur du même process, utilisez ces commandes pour effectuer les actions correspondantes (c'est-à-dire simplement assigner l'objet source à l'objet de destination).
• Si le glisser-déposer est interprocess, soyez vigilant lorsque vous accédez aux données glissées : vous devez récupérer l'instance des données située dans le process source. Si les données glissées proviennent d'une variable, utilisez la commande LIRE VARIABLE PROCESS pour obtenir la valeur correcte. Si les données glissées proviennent d'un champ, il est probable que l'enregistrement courant de la table n'est pas le même d'un process à l'autre, vous devez donc accéder au bon enregistrement.
Dans ce dernier cas, plusieurs solutions sont envisageables :
- Puisque l'événement Sur glisser de la méthode de l'objet de destination est exécuté dans le contexte du process source, vous pouvez copier, au moment de son exécution, les données du champ ou le numéro de l'enregistrement dans une variable interprocess, que vous réutiliserez lors de l'événement Sur déposer.

- Pendant l'événement Sur déposer, vous pouvez établir une communication interprocess avec le process source dans le but de récupérer les données requises.

Lorsque le glisser-déposer n'est pas destiné à déplacer des données mais plutôt à créer des métaphores d'interface pour des opérations particulières, vous pouvez virtuellement réaliser tout ce que vous voulez.

Les commandes du thème Conteneur de données

Si le glisser-déposer implique le déplacement de données hétérogènes ou de documents entre deux applications 4D ou une application 4D et une application tierce, les commandes du thème “Conteneur de données” vous fourniront tous les outils nécessaires.
En effet, ces commandes permettent de gérer à la fois le copier-coller et le glisser-déposer de données. 4D exploite deux conteneurs de données : l’un pour les données copiées (ou coupées), qui est en fait celui du Presse-papiers et l’autre pour les données en cours de glisser-déposer. Ces deux conteneurs sont gérés à l’aide des mêmes commandes. Vous accédez à l’un ou à l’autre en fonction du contexte.

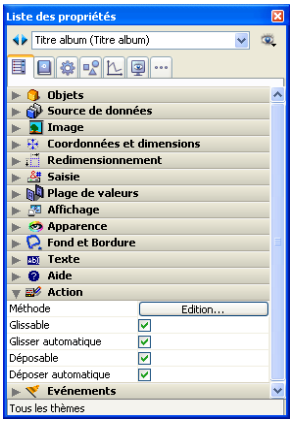
Pour plus d'informations sur l'utilisation des commandes du thème “Conteneur de données” dans le cadre du glisser-déposer, reportez-vous à la section Gestion du conteneur de données.

Glisser-Déposer automatique

Les objets texte (champs, variables, combo box et list box) ainsi que les objets image autorisent le glisser-déposer automatique, c’est-à-dire le déplacement ou la copie direct(e) de la sélection de texte ou d'une image d’une zone à une autre par simple clic. Il peut être employé dans la même zone 4D, entre deux zones 4D, ou entre 4D et une autre application, par exemple WordPad.

Note : En cas de glisser-déposer automatique entre deux zones 4D, les données sont *déplacées*, c’est-à-dire qu’elles sont supprimées de la zone source. Si vous souhaitez *recopier* les données, appuyez sur la touche **Ctrl** (Windows) ou **Commande** (Mac OS) pendant l’opération.

Le glisser-déposer automatique peut être paramétré séparément pour chaque objet d'un formulaire via deux options de la Liste des propriétés : **Glisser automatique** et **Déposer automatique** :



• **Glisser automatique** : Lorsque cette option est cochée, le mode de glisser automatique est activé pour l'objet. Ce mode est prioritaire pour les images, même si l'option **Glissable** est cochée. Dans ce mode, l'événement formulaire Sur début glisser n'est PAS généré.
Si vous souhaitez "forcer" l'utilisation du glisser standard, appuyez sur la touche **Alt** (Windows) ou **Option** (Mac OS) pendant l'opération.

• **Déposer automatique** : Cette option permet d'activer le mode de déposer automatique. Dans ce mode, 4D gère automatiquement — si possible — l'insertion des données glissées de type texte ou image et déposées sur l'objet (les données sont collées dans l'objet). Les événements Sur glisser et Sur déposer dans ce cas ne sont pas générés. En revanche, les événements Sur après modification (lors du déposer) et Sur données modifiées (lorsque l'objet perd le focus) sont générés.
En cas de déposer de données autres que du texte ou des images (autre objet 4D, fichier, etc.) ou de données complexes, l'application se réfère à la valeur de l'option **Déposable** : si elle est cochée, les événements Sur glisser et Sur déposer sont générés, dans le cas contraire le déposer est refusé. Ce principe dépend également de la valeur de l'option "Interdire de glisser des données ne provenant pas de 4D"(cf. ci-dessous).

Interdire de glisser des données ne provenant pas de 4D (compatibilité)

A compter de la version 11, 4D permet le glisser-déposer de sélections, d'objets ou de fichiers extérieurs à 4D, comme par exemple des fichiers image. Cette possibilité doit être prise en charge par le code de la base.

Dans les bases de données converties depuis une version précédente de 4D, cette possibilité peut entraîner des dysfonctionnements si le code existant n'est pas adapté. Pour cette raison, une option de Préférences permet d'inactiver cette fonction : **Interdire de glisser des données ne provenant pas de 4D**. Cette option est placée dans la page Application/Compatibilité. Elle est cochée par défaut dans les bases converties.

Lorsque cette option est cochée, le déposer d'objets externes est refusé dans les formulaires 4D. A noter toutefois que l'insertion d'objets externes reste possible dans les objets disposant de l'option **Déposer automatique**, lorsque l'application peut interpréter les données déposées (texte ou image).

Référence

Evenement formulaire, LIRE VARIABLE PROCESS, Liste existante, Position deposer, PROPRIETES GLISSER DEPOSER, RESOUDRE POINTEUR, Type.

Position deposer

Position deposer {(numColonne)} → Numérique

Paramètre	Type		Description
numColonne	Entier long	←	Numéro de colonne de list box ou -1 si le déposer a lieu après la dernière colonne
Résultat	Numérique	←	• Numéro (tableau/list box) ou • Position (liste hiérarchique) ou • Position dans la chaîne (texte/combo box) de l'élément de destination ou -1 si le déposer a lieu après le dernier élément de tableau ou de liste

Description

Position deposer permet de connaître l'emplacement, dans un objet de destination “complexe”, auquel un objet a été (glissé et) déposé. Généralement, vous utiliserez Position deposer pendant le traitement d'un événement glisser-déposer qui s'est produit dans un tableau, une list box, une liste hiérarchique ou un champ texte.

• Si l'objet de destination est un tableau, la fonction retourne un numéro d'élément.
• Si l'objet de destination est une list box, la fonction retourne un numéro de ligne. Dans ce cas, la fonction retourne également dans le paramètre facultatif numColonne le numéro de la colonne sur laquelle le déposer a eu lieu.
• Si l'objet de destination est une liste hiérarchique, la fonction retourne une position d'élément.
• Si l'objet de destination est une variable ou un champ de type texte ou encore une combo box, la fonction retourne une position de caractère à l'intérieur de la chaîne.
Dans tous les cas, la fonction retourne -1 si l'objet source a été déposé après le dernier élément de l'objet de destination.

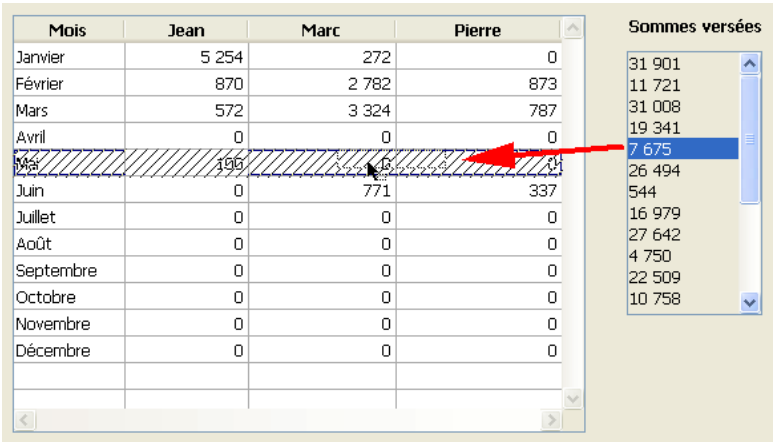
Si vous appelez Position deposer pendant le traitement d'un événement qui n'est pas de type glisser-déposer dans un tableau, une list box, une combo box, une liste hiérarchique ou un texte, la fonction retourne également -1.

Rappel : Pour qu'un objet de formulaire accepte des données déposées, la propriété **Déposable** doit lui avoir été assignée. De plus, sa méthode objet doit être appelée par l'événement Sur glisser et/ou Sur déposer si vous voulez pouvoir gérer ce type d'événement.

Exemples

(1) Reportez-vous aux exemples de la commande PROPRIETES GLISSER DEPOSER.

(2) Dans l'exemple suivant, une liste de sommes doit être ventilée par mois et par personne. L'opération s'effectue par glisser-déposer depuis une zone de défilement :



La méthode objet de la list box contient le code suivant :

```
Au cas ou
:(Evenement formulaire=Sur glisser )
    PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
    Si ($source=Pointeur vers("ZD1")) `Si le déposer provient bien de la zone de défilement
        $0:=0
    Sinon
        $0:=-1 `On refuse le déposer
    Fin de si
:(Evenement formulaire=Sur déposer )
    PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
    $numligne:=Position deposer($numcol)
    Si ($numcol=1)
        BEEP
```

```
        Sinon
            Au cas ou  `Addition des valeurs déposées
            : ($numcol=2)
                Jean{$numligne}:=Jean{$numligne}+ZD1{$lignetab}
            : ($numcol=3)
                Marc{$numligne}:=Marc{$numligne}+ZD1{$lignetab}
            : ($numcol=4)
                Pierre{$numligne}:=Pierre{$numligne}+ZD1{$lignetab}
            Fin de cas
            SUPPRIMER DANS TABLEAU(ZD1;$lignetab) `Mise à jour de la zone
        Fin de si
    Fin de cas
```

Référence

Présentation du Glisser-Déposer, PROPRIETES GLISSER DEPOSER.

PROPRIETES GLISSER DEPOSER (srcObjet; srcElément; srcProcess)

Paramètre	Type		Description
srcObjet	Pointeur	←	Pointeur vers l'objet source du glisser-déposer
srcElément	Numérique	←	Numéro de l'élément de tableau glissé ou Numéro de la ligne de list box glissée ou Elément de la liste hiérarchique glissé ou -1 si l'objet glissé n'est ni un élément de tableau, ni une ligne de list box ni un élement de liste
srcProcess	Numérique	←	Numéro du process source

Description

La commande PROPRIETES GLISSER DEPOSER vous permet de récupérer des informations sur l'objet source lorsque l'événement Sur glisser ou Sur déposer est déclenché pour un objet "complexe" (tableau, list box ou liste hiérarchique).

Généralement, la commande PROPRIETES GLISSER DEPOSER se place dans la méthode objet (ou une des sous-méthodes qu'elle appelle) de l'objet pour lequel l'événement Sur glisser ou Sur déposer se produit.

Rappel : Des données peuvent être déposées sur un objet de formulaire si la propriété **Déposable** lui a été assignée. De plus, la méthode qui lui est associée doit être appelée par l'événement Sur déposer et/ou Sur glisser si vous voulez traiter ce type d'événement.

Après l'appel de cette commande :

• Le paramètre srcObjet est un pointeur vers l'objet source, c'est-à-dire l'objet qui a été glissé et déposé. Notez que cet objet peut être ou non identique à l'objet de destination, autrement dit l'objet pour lequel l'événement Sur déposer ou Sur glisser a été déclenché. Le glisser-déposer de valeurs entre des objets de même type est utile pour les tableaux et les listes hiérarchiques : cela vous fournit un moyen simple de permettre à l'utilisateur de trier manuellement un tableau ou une énumération.

• Si les données glissées-déposées sont un élément de tableau (l'objet source étant un tableau), le paramètre srcElément est égal au numéro de cet élément. Si les données glissées-déposées sont une ligne de list box, le paramètre srcElément est égal au numéro de cette ligne. Si les données glissées-déposées sont un élément de liste hiérarchique, le paramètre srcElément

retourne la position de cet élément. Sinon, si l'objet source n'appartient à aucune de ces catégories, srcElément est égal à -1.

• Des opérations de glisser-déposer peuvent être effectuées entre différents process. Le paramètre srcProcess est égal au numéro du process auquel appartient l'objet source. Il est important de tester la valeur de ce paramètre. En effet, vous pouvez traiter un glisser-déposer “process” en copiant simplement les données source dans l'objet de destination. En revanche, lorsque vous traitez un glisser-déposer “interprocess”, vous devez utiliser la commande LIRE VARIABLE PROCESS pour récupérer les données source à partir de l'instance de l'objet du process source. Si l'objet source est un champ, vous devez récupérer sa valeur dans le process source via les outils de communication interprocess ou traiter ce cas particulier pendant que vous répondez à l'événement Sur glisser (reportez-vous aux paragraphes ci-dessous). Notez cependant que généralement, le glisser-déposer dans une interface utilisateur s'effectue à partir de variables (tableaux et liste hiérarchiques) vers des zones de saisie de données (champs ou variables).

Si vous appelez PROPRIETES GLISSER DEPOSER alors qu'aucun événement glisser-déposer ne s'est produit, srcObjet retourne un pointeur NIL, srcElément retourne -1 et srcProcess retourne 0.

Astuce : 4D gère pour vous l'aspect graphique du glisser-déposer. Mais c'est à vous de traiter l'événement de manière appropriée. Dans les exemples ci-dessous, le traitement consiste à copier les données qui ont été glissées. Mais vous pouvez également implémenter des interfaces plus sophistiquées dans lesquelles, par exemple, le glisser-déposer d'un élément de tableau depuis une palette flottante provoque le remplissage de la fenêtre de destination (la fenêtre dans laquelle se trouve l'objet de destination) avec des données structurées (comme plusieurs champs provenant d'un enregistrement désigné par l'élément de tableau source).

Vous pouvez appeler la commande PROPRIETES GLISSER DEPOSER lors de l'événement formulaire Sur glisser afin de décider si l'objet de destination doit ou non accepter l'opération, en fonction du type et/ou de la nature de l'objet source (ou pour toute autre raison). Si vous acceptez le glisser-déposer, la méthode de l'objet doit retourner $0:=0. Si vous n'acceptez pas l'opération, la méthode de l'objet doit retourner $0:=-1. L'acceptation ou le refus d'un glisser-déposer est visible à l'écran : l'objet sera ou ne sera pas sélectionnable (par exemple encadré) en tant que destination du glisser-déposer.

Astuce : Pendant l'événement Sur glisser, la méthode de l'objet de destination est exceptionnellement exécutée dans le contexte du process de l'objet source. Si le glisser-déposer est interprocess et si l'objet source est un champ, vous pouvez profiter de l'occasion pour copier les données source dans une variable interprocess. Ainsi, vous n'aurez pas besoin par la suite, pendant l'événement Sur déposer, d'ouvrir une communication interprocess avec le process source pour récupérer la valeur du champ glissé. Si l'objet source du glisser-déposer interprocess est une variable, vous pouvez utiliser la commande LIRE VARIABLE PROCESS pendant l'événement Sur déposer.

Exemples

(1) Vous disposez, dans plusieurs formulaires de votre base, de zones de défilement. Vous voulez que l'utilisateur puisse réordonner manuellement les éléments des zones par simple glisser-déposer à l'intérieur de chaque zone. Plutôt que d'écrire du code spécifique pour chaque cas, vous voulez utiliser une méthode projet générique qui traite toutes les zones de défilement. Pour cela, vous pouvez écrire :

```
` Méthode projet de traitement de glisser-déposer interne dans un tableau
` Traitement de glisser-déposer interne dans un tableau ( Pointeur ) -> Booléen
` Traitement de glisser-déposer interne dans un tableau ( -> Tableau ) -> Est un glisser-
                                                                    déposer interne dans un tableau

Au cas ou
   : (Evenement formulaire=Sur glisser)
      PROPRIETES GLISSER DEPOSER ($vpSrcObj;$vlSrcElem;$vlPID)
      Si ($vpSrcObj=$1)
            `Accepter le glisser-déposer s'il est interne au tableau
         $0:=0
      Sinon
         $0:=-1
      Fin de si
   : (Evenement formulaire=Sur déposer)
            ` Récupérer les informations sur l'objet source du glisser-déposer
      PROPRIETES GLISSER DEPOSER ($vpSrcObj;$vlSrcElem;$vlPID)
         ` Récupérer le numéro de l'élément de destination
      $vlDstElem := Position deposer
         ` Si l'élément n'a pas été glissé-déposé sur lui-même
      Si ($vlDstElem # $vlSrcElem)
            ` Stocker l'élément glissé dans l'élément 0 du tableau
         $1->{0}:=$1->{$vlSrcElem}
            ` Effacer l'élément glissé
         SUPPRIMER DANS TABLEAU ($1->;$vlSrcElem)
            ` Si l'élément de destination est au-delà de l'élément glissé
         Si ($vlDstElem>$vlSrcElem)
               ` Décrémenter le numéro de l'élément de destination
            $vlDstElem:=$vlDstElem-1
         Fin de si
               ` Si le glisser-déposer s'est produit au-delà du dernier élément
         Si ($vlDstElem=-1)
               ` Définir le numéro de l'élément de destination comme un nouvel
                                                                  ` élément ajouté à la fin du
tableau
```

```
            $vlDstElem:=Taille tableau ($1->)+1
        Fin de si
            ` Insérer ce nouvel élément
        INSERER DANS TABLEAU ($1->;$vlDstElem)
            ` Fixer sa valeur, préalablement stockée dans l'élément zéro du tableau
        $1->{$vlDstElem}:=$1->{0}
            ` L'élément devient le nouvel élément sélectionné du tableau
        $1->:=$vlDstElem
    Fin de si
Fin de cas
```

Une fois que vous avez écrit cette méthode projet, vous pouvez l'utiliser ainsi :

```
    ` Méthode objet Zone de défilement unTableau

Au cas ou
        `...
    : (Evenement formulaire=Sur glisser )
        $0:=Traitement de glisser-déposer interne dans un tableau (Self)
    : (Evenement formulaire=Sur déposer )
        Traitement de glisser-déposer interne dans un tableau (Self))
            `...
Fin de cas
```

(2) Vous disposez, dans plusieurs formulaires de votre base, de zones de texte saisissables. Vous voulez que l'utilisateur puisse y saisir des données par glisser-déposer à partir de sources multiples. Plutôt que d'écrire du code spécifique pour chaque cas, vous voulez utiliser une méthode projet générique qui traite toutes les zones de texte. Pour cela, écrivez la méthode suivante :

```
    ` Méthode projet Traitement du déposer dans variable Texte
    ` Traitement du déposer dans variable Texte ( Pointeur )
    ` Traitement du déposer dans variable Texte ( -> variable texte ou chaîne )

Au cas ou
    ` Utilisation de cet événement pour accepter ou refuser le glisser-déposer
    : (Evenement formulaire=Sur glisser)
            ` Initialiser $0 pour le refus
        $0:=-1
            ` Récupérer les informations sur l'objet source du glisser-déposer
        PROPRIETES GLISSER DEPOSER($vpSrcObj;$vlSrcElem;$vlPID)
                                        ` Dans cet exemple, nous refusons le glisser-
déposer d'un objet sur lui-même
```

```
Si ($vpSrcObj#$1)
            ` Récupérer le type des données glissées
    $vlSrcType:=Type($vpSrcObj->)
    Au cas ou
        : ($vlSrcType=Est un champ alpha)
                ` OK pour les champs alphanumériques
            $0:=0
                ` Copie immédiate de la valeur dans une variable interprocess
            <>vtDonnéesGlissées:=$vpSrcObj->
        : ($vlSrcType=Est un texte)
                ` OK pour les champs ou variables texte
            $0:=0
            RESOUDRE POINTEUR($vpSrcObj;$vsVarName;$vlTableNum;
                                                        $vlFieldNum)
                ` Si c'est un champ
            Si (($vlTableNum>0) & ($vlFieldNum>0))
                ` Copie immédiate de la valeur dans une variable interprocess
                <>vtDonnéesGlissées:=$vpSrcObj->
            Fin de si
        : ($vlSrcType=Est une variable chaîne)
                ` OK pour les variables chaîne
            $0:=0
        : (($vlSrcType=Est un tableau chaîne) | ($vlSrcType=Est un tableau texte))
                ` OK pour les tableaux chaîne et texte
            $0:=0
        : (($vlSrcType=Est un entier long) | ($vlSrcType=Est un numérique))
            Si (Liste existante($vpSrcObj->))
                ` OK pour les liste hiérarchiques
                $0:=0
            Fin de si
    Fin de cas
Fin de si

` Utilisation de cet événement pour effectuer réellement l'action de glisser-déposer
: (Evenement formulaire=Sur déposer)
    $vtDonnéesGlissées:=""
        ` Récupérer les informations sur l'objet source du glisser-déposer
    PROPRIETES GLISSER DEPOSER($vpSrcObj;$vlSrcElem;$vlPID)
    RESOUDRE POINTEUR($vpSrcObj;$vsVarName;$vlTableNum;$vlFieldNum)
```

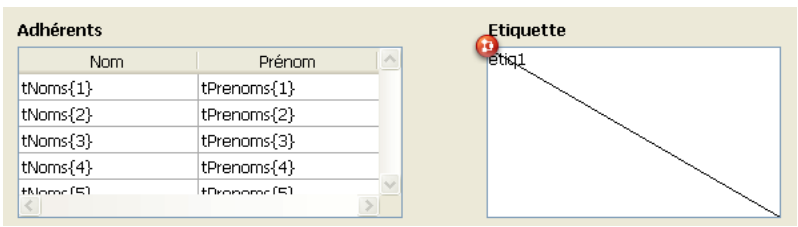
```
    `Si c'est un champ
Si (($vlTableNum>0) & ($vlFieldNum>0))
        ` Récupérons la variable interprocess créée lors du Sur glisser
    $vtDonnéesGlissées:=<>vtDonnéesGlissées
Sinon
        ` Récupérer le type des données glissées
    $vlSrcType:=Type($vpSrcObj->)
    Au cas ou
            ` Si c'est un tableau
        : (( $vlSrcType=Tableau chaîne) | ($vlSrcType=Tableau texte))
            Si ($vlPID#Numero du process courant)
            ` Lire l'élément depuis l'instance de la variable dans le process source
                LIRE VARIABLE PROCESS ($vlPID;$vpSrcObj->{$vlSrcElem};
                                                        $vtDraggedData)
            Sinon
                ` Glisser-déposer depuis le même process, copions juste la valeur
                $vtDraggedData:=$vpSrcObj->{$vlSrcElem}
            Fin de si
            ` Si c'est une liste
        : (($vlSrcType=Est un numérique ) | ($vlSrcType=Est un entier long ))
                ` Si c'est une liste en provenance d'un autre  process
            Si ($vlPID#Numero du process courant)
                    `Récupérer la référence de la liste dans l'autre process
                LIRE VARIABLE PROCESS ($vlPID;$vpSrcObj->;$vlList)
            Sinon
                $vlList:=$vpSrcObj->
            Fin de si
                ` Si la liste existe
            Si (Liste existante($vpSrcObj->))
                    `Récupérer le texte de l'élément dont on a obtenu la position
                INFORMATION ELEMENT($vlList;$vlSrcElem;$vlItemRef;
                                                        $vsItemText)
                $vtDraggedData:=$vsItemText
            Fin de si
    Sinon
            ` C'est une variable chaîne ou texte
            Si ($vlPID#Numero du process courant)
                LIRE VARIABLE PROCESS ($vlPID;$vpSrcObj->;$vtDraggedData)
            Sinon
                $vtDraggedData:=$vpSrcObj->
                                        Fin de si
```

```
            Fin de cas
        Fin de si
        ` S'il y a effectivement quelque chose à déposer (l'objet source pourrait être vide)
        Si ($vtDraggedData # "")
        ` Vérifions que la longueur de la variable texte ne dépasse pas 32 000 caractères
            Si ((Longueur($1->)+Longueur($vtDraggedData))<=32000)
                $1->:=$1->+$vtDraggedData
            Sinon
                BEEP
                ALERTE("Le glisser-déposer ne peut être effectué car il y aurait trop de
                                                                                texte.")
            Fin de si
        Fin de si
Fin de cas
```

Une fois que vous avez écrit cette méthode projet, vous pouvez l'utiliser ainsi :

```
` Méthode objet du champ de texte [uneTable]unTexte

Au cas ou
    ` ...
: (Evenement formulaire=Sur glisser )
    $0:=Traitement du déposer dans variable texte  (Self)

: (Evenement formulaire=Sur déposer )
    Traitement du déposer dans variable texte  (Self)
    ` ...

Fin de cas
```

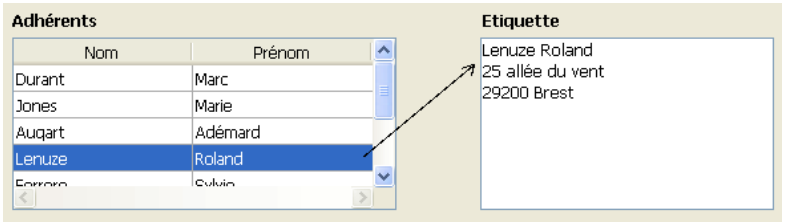
(3) Nous souhaitons remplir une zone de texte (par exemple une étiquette) avec des données glissées depuis une list box.



Voici la méthode de l'objet etiq1 :

```
Au cas ou
:(Evenement formulaire=Sur glisser )
    PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
    Si ($source=Pointeur vers("list box1"))
        $0:=0   `On accepte le glisser
    Sinon
        $0:=-1 `On refuse le glisser
    Fin de si
:(Evenement formulaire=Sur déposer )
    PROPRIETES GLISSER DEPOSER($source;$lignetab;$numprocess)
    CHERCHER([Adhérents];[Adhérents]Nom=tNoms{$lignetab})
    Si (Enregistrements trouves([Adhérents])#0)
        etiq1:=[Adhérents]Nom+" "+[Adhérents]Prénom+Caractere(Retour chariot )+
                                                    [Adhérents]Adresse+Caractere(Retour chariot )+
                                                        [Adhérents]Code postal+" "+[Adhérents]Ville
    Fin de si
Fin de cas
```

Il est dès lors possible d'effectuer l'action suivante :



Référence

Evenement formulaire, LIRE VARIABLE PROCESS, Liste existante, Position deposer, Présentation du Glisser-Déposer, RESOUDRE POINTEUR.

Méthode base Sur déposer

La Méthode base Sur déposer est disponible dans les applications 4D locales ou distantes.

Cette méthode base est exécutée automatiquement en cas de déposer d'objets dans l'application 4D en-dehors de tout formulaire ou fenêtre, c'est-à-dire :
• dans une zone vide de la fenêtre MDI (Windows),
• sur l'icône 4D dans le Dock (Mac OS) ou sur le Bureau du système.

Sous Mac OS, il est nécessaire de maintenir les touches **Option**+**Commande** enfoncées pendant le déposer afin que la méthode base soit appelée.

Lors d'un déposer sur l'icône de l'application 4D sur le bureau, la Méthode base Sur déposer est appelée uniquement si l'application est déjà lancée, sauf dans le cas des applications fusionnées avec 4D Desktop. Dans ce cas, la méthode base est appelée même lorsqu'elle n'est pas lancée. Ce principe permet de définir des signatures de documents personnalisées.

Exemple

Cet exemple permet d'ouvrir un document 4D Write déposé en-dehors de tout formulaire :

```
    `Méthode base Sur Déposer
fichierDéposé:=Lire fichier dans conteneur(1)
Si(Position(".4W7";fichierDéposé)=Longueur(fichierDéposé)-3)
    zexterne:=Creer fenetre externe(100;100;500;500;0;fichierDéposé;"_4D Write")
    WR OUVRIR DOCUMENT(zexterne;fichierDéposé)
Fin de si
```

Référence

Présentation des méthodes base.

28

Graphes

GRAPHE (zoneGraphe; graphNum; xCatégories; zValeurs{; zValeurs2; ...; zValeursN})

Paramètre	Type		Description
zoneGraphe	Graphe | Var image	→	Zone de graphe ou Variable image
graphNum	Numérique	→	Numéro de type de graphe
xCatégories	Tableau	→	Catégories sur l'axe des x
zValeurs	Tableau	→	Valeurs à représenter graphiquement (jusqu'à 8 valeurs)

Description

La commande GRAPHE crée un graphe dans une zone de graphe ou une variable image placée dans un formulaire à partir de valeur provenant de tableaux. La commande GRAPHE doit impérativement être placée dans la méthode formulaire ou dans une méthode objet appartenant au formulaire, ou encore dans une méthode projet appelée par l'une des deux précédentes.

Les graphes générés par cette commande peuvent être dessinés soit à l'aide du plug-in intégré 4D Chart, soit, depuis la version 11 de 4D, via le moteur de rendu SVG intégré.

Note : SVG (*Scalable Vector Graphics*) est un format de fichier graphique vectoriel (extension .svg). Basé sur le XML, ce format est largement répandu et peut être notamment affiché par les navigateurs Web. Pour plus d'informations, reportez-vous à l'adresse http://www.w3.org.Graphics/SVG/. La commande SVG EXPORTER VERS IMAGE vous permet également de tirer parti du moteur SVG intégré.

C'est le type du paramètre zoneGraphe qui détermine le moteur graphique utilisé pour le rendu : si vous passez une référence de zone 4D Chart ou une variable de zone de graphe, le plug-in 4D Chart sera utilisé. Si vous passez une variable image, le moteur SVG sera utilisé. Vous pouvez choisir le type de moteur en fonction des critères suivants :
• Les graphes générés par 4D Chart peuvent être entièrement contrôlés, manipulés et enrichis par programmation, via les commandes du plug-in 4D Chart. Pour plus d'informations sur les commandes de 4D Chart, reportez-vous au manuel *Langage* de 4D Chart.
• Les graphes générés par le moteur SVG ont un aspect plus moderne et bénéficient des fonctions d'interface associées aux variables images : menu contextuel en mode Application (permettant notamment le choix du format d'affichage), barres de défilement, etc.

Passez dans le paramètre zoneGraphe soit un nom de zone de graphe (ou une référence de zone 4D Chart), soit une variable image 4D, en fonction du moteur de rendu à utiliser. Ces zones sont créées en mode Développement, dans l'éditeur de formulaires. Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Le paramètre graphNum définit le type de graphe à utiliser. Vous devez passer un nombre entre 1 et 8. Les différents types de graphes disponibles sont listés dans l'exemple présenté plus bas. Une fois le graphe créé, vous pouvez modifier son type en modifiant la valeur de graphNum et en exécutant de nouveau la commande GRAPHE.

Le paramètre xCatégories définit les valeurs qui seront utilisées pour l'axe des X. Vous pouvez passer un champ de type chaîne, Heure, Date, ou un type numérique. Il doit y avoir le même nombre d'éléments de tableau dans xCatégories qu'il y en a dans chaque zValeurs.

Le paramètre zValeurs définit les valeurs à représenter graphiquement. Elles doivent être de type numérique. Vous pouvez passer jusqu'à huit ensembles de données. Les graphes en secteurs ne représentent que le premier zValeurs.

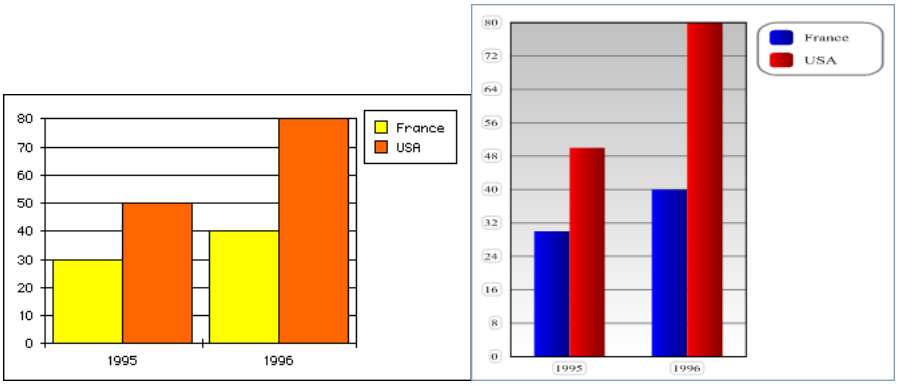
Exemple

L'exemple suivant illustre les différents types de graphes que vous pouvez obtenir avec chaque moteur graphique. Ce code doit être placé dans la méthode formulaire (ou une méthode objet) du formulaire contenant la zone de graphe ou la variable image. A noter que, dans notre exemple, les données représentées sont constantes, ce qui n'est généralement pas le cas :

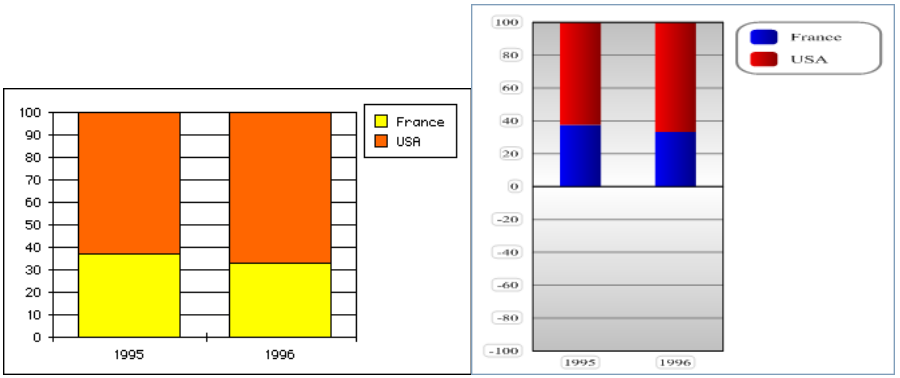
```
C_IMAGE (vGraph)  `A passer si vous souhaitez utiliser le moteur SVG
TABLEAU ALPHA (4; X; 2)  ` Création d'un tableau pour l'axe des X
X{1} := "1995"  ` libellé X #1
X{2} := "1996"  ` Libellé X #2
TABLEAU REEL (A; 2)  ` Création d'un tableau pour l'axe des Z
A{1} := 30  ` Insertion des données
A{2} := 40
TABLEAU REEL (B; 2) ` Création d'un second tableau pour l'axe des Z
B{1} := 50  ` Insertion des données
B{2} := 80
GRAPHE (vGraph; vType; X; A; B) ` Dessiner le graphe
    ` Définition des légendes du graphe
PARAMETRES DU GRAPHE (vGraph; 0; 0; 0; 0; Faux; Faux; Vrai; "France"; "USA")
```

Les images suivantes représentent les graphes résultants avec chaque moteur de rendu (4D Chart puis SVG) :

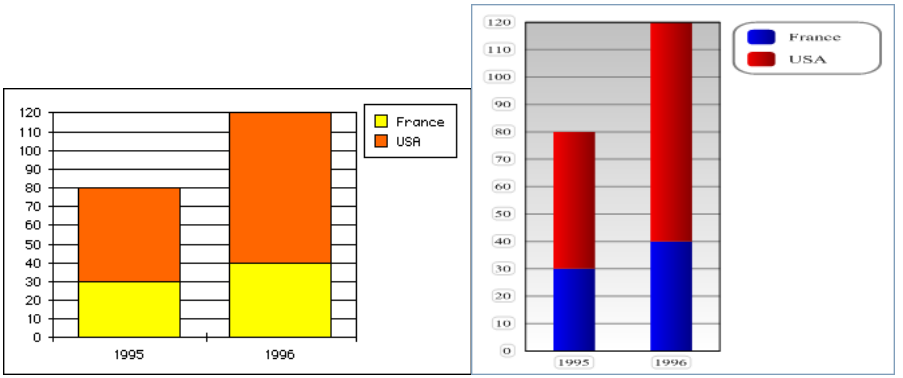
• Lorsque vType est égal à 1, vous obtenez un graphe en **Colonnes** :



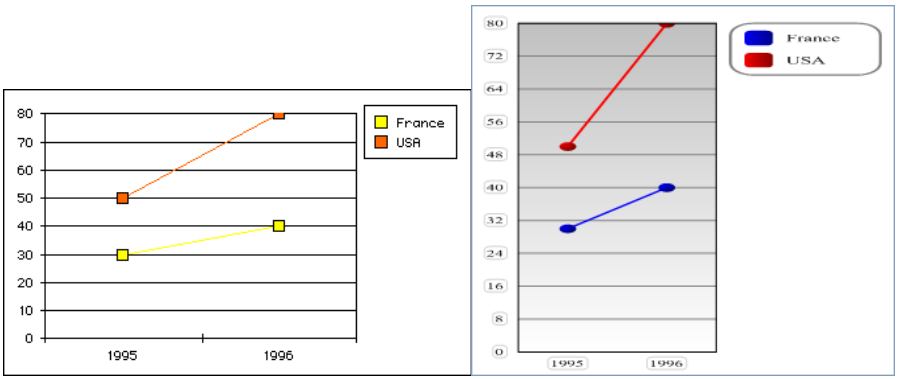
• Lorsque vType est égal à 2, vous obtenez un graphe en **Colonnes proportionnelles** :



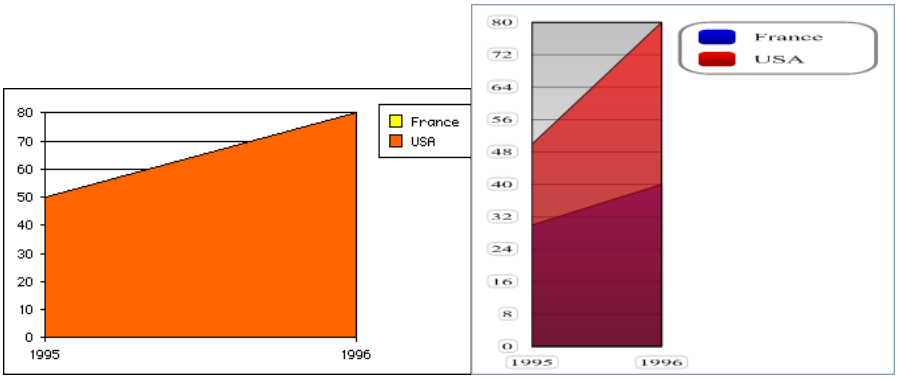
• Lorsque vType est égal à 3, vous obtenez un graphe en **Colonnes empilées** :



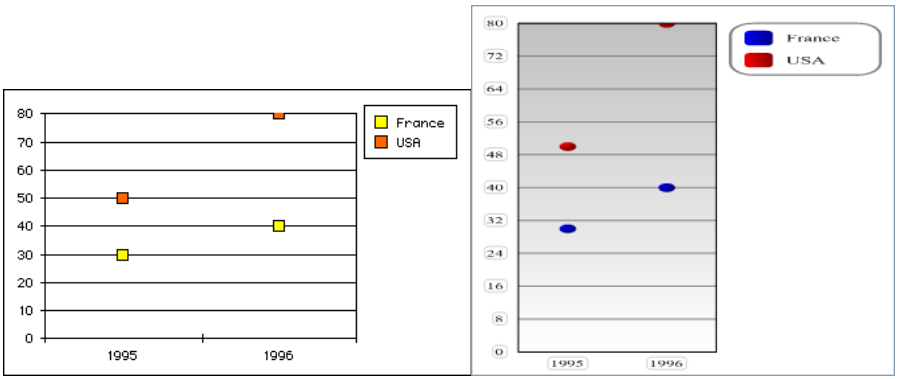
• Lorsque vType est égal à 4, vous obtenez un graphe en **Lignes** :



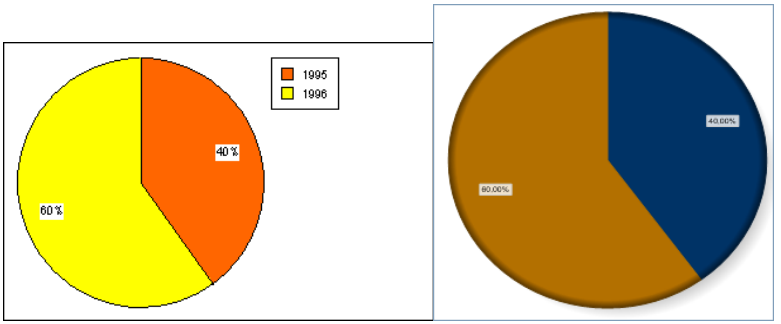
• Lorsque vType est égal à 5, vous obtenez un graphe en **Aires** :



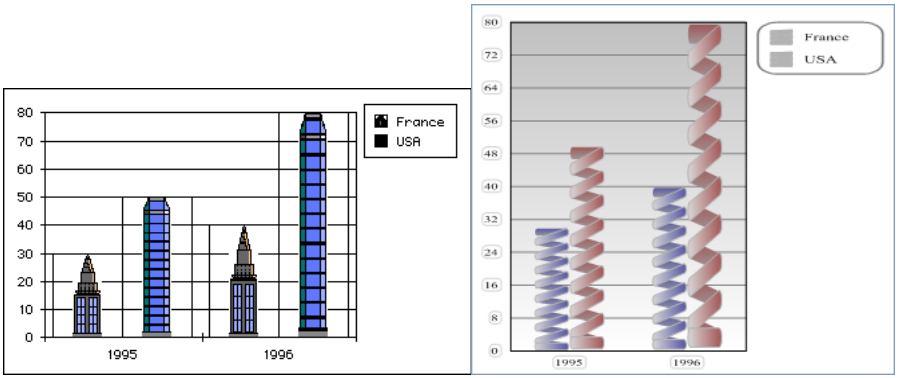
• Lorsque vType est égal à 6, vous obtenez un graphe en **Points** :



• Lorsque vType est égal à 7, vous obtenez un graphe en **Secteurs** :



• Lorsque vType est égal à 8, vous obtenez un graphe en **Images** :



Référence

ch_Donnees vers graphe, ch_Selection vers graphe, ch_Tableaux vers graphe, GRAPHE SUR SELECTION, PARAMETRES DU GRAPHE, SVG EXPORTER VERS IMAGE.

GRAPHE SUR SELECTION

version 6.0 (Modifiée)

GRAPHE SUR SELECTION{(table)}

ou :

GRAPHE SUR SELECTION ({table; }numGraphe; axeX; axeZ{; axeZ2; ...; axeZN})

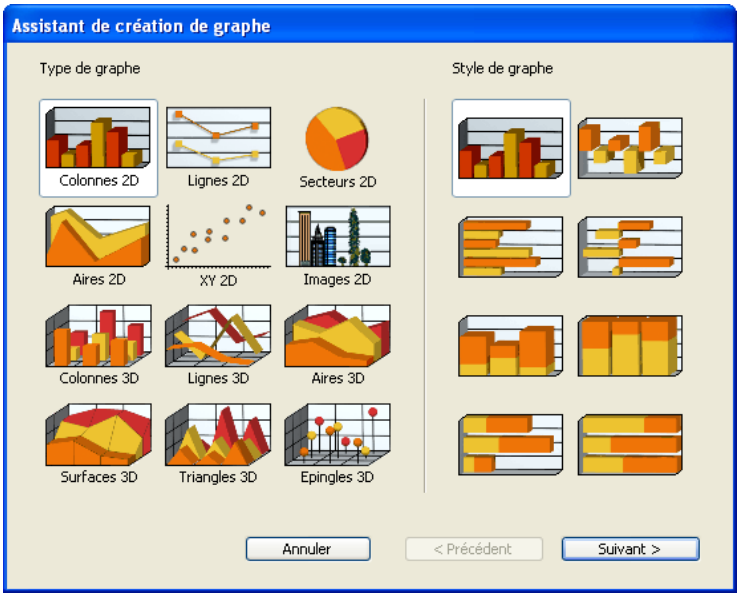
Paramètre	Type		Description
table	Table	→	Table de la sélection à représenter graphiquement ou Table par défaut si ce paramètre est omis
numGraphe	Numérique	→	Numéro de type de graphe
axeX	Champ	→	Valeurs sur l'axe des X
axeZ	Champ	→	Champ(s) à représenter (jusqu'à 8 champs)

Description

GRAPHE SUR SELECTION accepte deux syntaxes. La première syntaxe fait apparaître l'Assistant de création de graphe de 4D Chart, permettant à l'utilisateur de sélectionner les champs à représenter graphiquement. Dans la seconde syntaxe, vous désignez les champs à représenter et l'Assistant de création de graphe ne s'affiche pas.

GRAPHE SUR SELECTION ne traite que les valeurs des champs d'une table. Seules les valeurs de la sélection courante du process courant sont représentées.

La première syntaxe est équivalente à la sélection de la commande **Graphes...** dans le menu **Outils**, en mode Développement. L'écran suivant représente la première page de l'assistant de création de graphe, permettant à l'utilisateur de choisir un type de graphe.



Pour plus d'informations sur l'emploi de cet assistant, reportez-vous au manuel *Mode Développement*.

La seconde syntaxe de la commande représente graphiquement les champs spécifiés de table.

Le paramètre numGraphe définit le type de graphe à utiliser. Vous devez passer un nombre entre 1 et 8. Les différents types de graphes 4D Chart disponibles sont représentés dans l'exemple de la commande GRAPHE.

Note : La commande GRAPHE SUR SELECTION ne pemet pas d'utiliser le moteur de rendu SVG de 4D.

Le paramètre axeX définit les valeurs qui seront utilisées pour l'axe des X. Vous pouvez passer un champ de type Alpha, Entier, Entier long, Numérique ou Date.

Le paramètre axeZ définit les valeurs à représenter graphiquement. Vous pouvez passer un champ de type Entier, Entier long ou Numérique. Vous pouvez passer jusqu'à huit champs dans axeZ, séparés par deux points.

Quelle que soit la syntaxe employée, GRAPHE SUR SELECTION crée une fenêtre 4D Chart dans laquelle vous travaillez avec les graphes que vous créez.

Note : Vous pouvez également créer des graphes avec 4D dans l'Editeur d'états rapides, en demandant une impression vers un graphe. Pour plus d'informations sur ce point, reportez-vous au manuel *Mode Développement* de 4D.

Exemples

(1) L'exemple suivant illustre l'utilisation de la première syntaxe de GRAPHE SUR SELECTION. La fenêtre de l'Assistant de création de graphe s'affiche, permettant à l'utilisateur de sélectionner un type de graphe. La première méthode permet à l'utilisateur d'effectuer une recherche dans la table [Personnes], puis d'effectuer un tri ; l'Assistant de création de graphes est ensuite affiché :

```
CHERCHER ([Personnes])
Si(OK=1)
    TRIER ([Personnes])
    Si(OK=1)
        GRAPHE SUR SELECTION([Personnes])
    Fin de si
Fin de si
```

(2) L'exemple suivant illustre l'utilisation de la seconde syntaxe de GRAPHE SUR SELECTION. Une recherche puis un tri sont d'abord effectués sur les enregistrements de la table [Personnes]. Les salaires des personnes sont alors représentés graphiquement :

```
CHERCHER ([Personnes]; [Personnes]Titre = "Directeur")
TRIER ([Personnes]; [Personnes]Salaire; >)
GRAPHE SUR SELECTION([Personnes]; 1; [Personnes]Noms; [Personnes]Salaire)
```

Référence

ch_Selection vers graphe, GRAPHE.

PARAMETRES DU GRAPHE

version 11 (Modifiée)

PARAMETRES DU GRAPHE (zoneGraphe; xmin; xmax; ymin; ymax; xprop; grilleX; grilleY; titre{; titre2; ...; titreN})

Paramètre	Type		Description
zoneGraphe	Graphe | Var image	→	Zone de graphe ou Variable image
xmin	Numérique, Date ou Heure	→	Valeur minimale de l'échelle des X pour graphe proportionnel (lignes ou points)
xmax	Numérique, Date ou Heure	→	Valeur maximale de l'échelle des X pour graphe proportionnel (lignes ou points)
ymin	Numérique	→	Valeur minimale de l'échelle des Y
ymax	Numérique	→	Valeur maximale de l'échelle des Y
xprop	Booléen	→	VRAI pour l'échelle des X proportionnelle ; FAUX pour l'échelle des X normale (lignes ou points)
grilleX	Booléen	→	VRAI pour la grille sur l'axe des X ; FAUX pour pas de grille sur l'axe des X (seulement si xprop est VRAI)
grilleY	Booléen	→	VRAI pour la grille sur l'axe des Y; FAUX pour pas de grille sur l'axe des Y
titre	Alphanumérique	→	Titre(s) pour les titre(s) des série(s)

Description

La commande PARAMETRES DU GRAPHE permet de paramétrer les échelles et les grilles d'un graphe placé dans un formulaire. Le graphe doit déjà être affiché à l'aide de la commande GRAPHE. Il peut avoir été dessiné via le plug-in 4D Chart (zoneGraphe est une variable graphe ou une référence de zone 4D Chart) ou le moteur SVG (zoneGraphe est un variable image). PARAMETRES DU GRAPHE ne fait rien s'il s'agit d'un graphe de type secteurs. Cette commande doit impérativement être placée dans la méthode formulaire ou dans une méthode objet appartenant au formulaire, ou encore dans une méthode projet appelée par l'une des deux précédentes.

Les paramètres xmin, xmax, ymin et ymax fixent les valeurs minimales et maximales pour les axes des X ou Y. Si la valeur des deux paramètres correspondants au même axe est nulle (0, ?00:00:00? ou !00/00/00! selon le type de données), les valeurs de graphe par défaut seront utilisées.

Le paramètre xprop fixe l'axe des X comme proportionnel (sont concernés par cette option les graphes de type 4 et 6). Lorsque ce paramètre est Vrai, chaque point sera mis sur l'axe des X par rapport aux valeurs des points si elles sont de type numérique, heure ou date.

Les paramètres grilleX et grilleY montrent ou cachent les grilles. Une grille pour l'axe des X sera affichée s'il s'agit d'un graphe en points ou en lignes proportionnel.

Le(s) paramètre(s) titre spécifient les titres des légendes.

Exemple

Reportez-vous à l'exemple de la commande GRAPHE.

Référence

GRAPHE, GRAPHE SUR SELECTION.

29

Images

Formats natifs pris en charge

4D intègre une gestion native des formats d'image les plus courants, tels que JPEG ou GIF. Cela signifie que les images sont affichées et stockées dans leur format d'origine, sans interprétation dans 4D. Les spécificités des différents formats (ombrages, zones transparentes...) sont conservées en cas de copier-coller et affichées sans altération. Cette prise en charge native est valide pour toutes les images stockées dans 4D : images de la bibliothèque, images collées dans les formulaires en mode Développement, images collées dans les champs ou variables en mode Application, etc.

Les formats natifs sont disponibles dans tous les cas et seront toujours retournés par la commande LISTE CODECS IMAGES, quels que soient le système d'exploitation et la configuration de la machine. Ces formats sont :
- Jpeg
- Png
- Bmp
- Gif
- Tif
- Emf (sous Windows uniquement)
- Pict
- Svg
- Pdf (sous Mac OS uniquement)

Note : Si 4D ne peut pas interpréter le format d'une image, le programme fait appel aux routines de Quicktime (cf. ci-dessous).

Identifiants de codecs d'images

Les formats d'images reconnus par 4D sont retournés par la commande LISTE CODECS IMAGES sous forme d'identifiants de codecs d'images. Ces identifiants peuvent être de trois formes :
- une extension (par exemple ".gif")
- un type Mime (par exemple "image/jpg")
- un code QuickTime sur 4 caractères (par exemple "PNTG")

La forme utilisée pour chaque format dépend du mode de déclaration du codec au niveau du système d'exploitation.
La plupart des commandes de gestion d'images de 4D attendent un identifiant de codec en paramètre. Il est donc impératif d'utiliser l'identifiant système retourné par la commande LISTE CODECS IMAGES.

Coordonnées d'un clic sur une image

4D permet de récupérer les coordonnées locales d'un clic dans un champ ou une variable image, même si un défilement ou un zoom a été appliqué à l'image.
Les coordonnées du clic sont retournées dans les Variables système MouseX et MouseY. Les coordonnées sont exprimées en pixels par rapport à l'angle supérieur gauche de l'image (0,0). Vous devez lire la valeur de ces variables dans le cadre de l'événement formulaire Sur clic ou Sur double clic. Pour que ce mécanisme fonctionne correctement, le format d'affichage doit être "Tronquée non centrée" (cf. commande CHOIX FORMATAGE).
Ce mécanisme, proche de celui d'une *image map*, peut être utilisé par exemple pour gérer des barres de boutons défilables ou l'interface de logiciels de cartographie.

Opérateurs sur les images

4D vous permet d'effectuer des **opérations** sur les images 4D, telles que la concaténation, la superposition, etc. Ce point est traité dans la section Opérateurs sur les images.

Utiliser QuickTime d'Apple avec 4D

4D peut s'appuyer sur QuickTime d'Apple pour gérer le stockage et l'affichage des images dans votre base de données.
Sous Mac OS, QuickTime est intégré au système d'exploitation, aucune extension n'est requise. Sous Windows, 4D requiert que la **version 4 minimum de QuickTime** soit installée pour que vous puissiez utiliser la compression/décompression d'images sur cette plate-forme.

Note de compatibilité : Les commandes CHARGER ET COMPRESSER IMAGE, COMPRESSER FICHIER IMAGE et COMPRESSER IMAGE font appel à des mécanismes obsolètes. Elles sont avantageusement remplacées par les commandes ECRIRE FICHIER IMAGE, IMAGE VERS BLOB ou CONVERTIR IMAGE. En outre, les commandes faisant appel à des fichiers disque (CHARGER ET COMPRESSER IMAGE et COMPRESSER FICHIER IMAGE) ne fonctionnent pas sous Windows, quelle que soit la version de QuickTime installée.

Erreurs de compression ou de conversion d'image

Le code d'erreur -9955 est retourné par 4D quand vous essayez d'utiliser une commande de compression ou de conversion d'image alors que QuickTime n'est pas installé dans votre système. D'autres erreurs générées par QuickTime peuvent être aussi retournées. Vous pouvez intercepter ces erreurs en utilisant une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

BLOB VERS IMAGE

version 11.2 (Modifiée)

BLOB VERS IMAGE (blobImage; image{; codec})

Paramètre	Type		Description
blobImage	BLOB	→	BLOB contenant une image
image	Image	←	Champ ou variable image 4D
codec	Chaîne	→	Identifiant de codec d'image

Description

La commande BLOB VERS IMAGE place dans un champ ou une variable image 4D une image stockée dans un BLOB, quel que soit son format initial.

Le fonctionnement de cette commande est analogue à celui de la commande LIRE FICHIER IMAGE ; elle s'applique simplement à un BLOB et non à un fichier. Elle permet d'afficher à tout moment des images stockées en format natif dans des BLOBs à l'aide, par exemple, de la commande DOCUMENT VERS BLOB ou IMAGE VERS BLOB.

Vous passez dans le paramètre blobImage le BLOB contenant l'image. L'image peut être de tout format pris en charge en natif par 4D ou compatible QuickTime. Vous pouvez obtenir la liste des formats disponibles à l'aide de la commande LISTE CODECS IMAGES. Si vous passez le paramètre facultatif codec, 4D utilisera la valeur fournie dans ce paramètre pour décoder le BLOB (voir ci-dessous le fonctionnement spécifique de la commande avec ce troisième paramètre).

Vous passez dans le paramètre image la variable ou le champ 4D de type image devant afficher l'image.

Note : Le format interne de l'image sera conservé au sein de la variable ou du champ 4D. Dans le cas de formats personnalisés utilisant QuickTime, il sera nécessaire de disposer de QuickTime pour afficher ultérieurement l'image dans 4D.

Après l'exécution de la commande, si le BLOB a pu être correctement décodé, l'image contient l'image affichable dans 4D.

Le paramètre facultatif codec vous permet de préciser le codec à utiliser pour décoder le BLOB.
Si vous passez dans codec un codec reconnu par 4D (retourné par la commande LISTE CODECS IMAGES), il est appliqué au BLOB et l'image est retournée dans le champ ou la variable image.

Si vous passez dans codec un codec non reconnu par 4D, un nouveau codec est enregistré dynamiquement avec l'identifiant passé en paramètre. 4D retourne alors une image qui encapsule le BLOB et la variable *OK* prend la valeur 1. Dans ce cas, pour récupérer le BLOB, il sera nécessaire d'utiliser la commande IMAGE VERS BLOB avec le même identifiant personnalisé. Ce mécanisme particulier permet de répondre à deux besoins spécifiques :
• encapsulation d'un BLOB (qui n'est pas une image) dans une image,
• chargement d'une image sans disposer du codec.
La mise en oeuvre de ces mécanismes permet de notamment de créer des "tableaux de BLOBs" en passant par des tableaux images. Cette technique doit être utilisée avec précaution car, les tableaux étant entièrement chargés en mémoire, la manipulation de BLOBs de grande taille peut altérer le fonctionnement de l'application.

Note : Un BLOB créé par la commande VARIABLE VERS BLOB est géré automatiquement, il n'est pas nécessaire de passer le codec pour l'encapsuler, le BLOB étant "signé". Pour l'opération inverse dans ce cas, vous devez passer ".4DVarBlob" comme identifiant de codec à la commande IMAGE VERS BLOB.

Référence

IMAGE VERS BLOB, LIRE FICHIER IMAGE, LISTE CODECS IMAGES, LISTE TYPES IMAGES.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système *OK* prend la valeur 1. En cas d'échec (absence de QuickTime, format d'image inconnu, paramètre codec reconnu mais ne validant pas le BLOB...), *OK* prend la valeur 0 et le champ ou la variable image 4D est retourné vide.

CHARGER ET COMPRESSER IMAGE

version 5

CHARGER ET COMPRESSER IMAGE (document; type; qualité; image)

Paramètre	Type		Description
document	DocRef	→	Numéro de référence du document
type	Alpha	→	Type de compression
qualité	Numérique	→	Qualité de compression (1...1000)
image	Image	←	Image compressée

Note de compatibilité : Cette commande fait appel à des mécanismes obsolètes et est conservée pour des raisons de compatibilité uniquement. Elle est avantageusement remplacée par les commandes LIRE FICHIER IMAGE et CONVERTIR IMAGE.

Description

CHARGER ET COMPRESSER IMAGE compresse une image chargée d'un document sur disque.

Note : Cette commande ne fonctionne pas sous Windows.

Vous pouvez ouvrir un document PICT à l'aide de la commande Ouvrir document et utiliser le numéro de référence retourné par cette fonction pour charger et compresser la PICT qu'il contient. Cette commande charge l'image en mémoire, la compresse en utilisant le type et la qualité que vous spécifiez et la retourne dans image.

L'image est chargée en mémoire avant d'être compressée. S'il n'y a pas assez de mémoire pour charger l'image, utilisez COMPRESSER FICHIER IMAGE avant d'appeler CHARGER ET COMPRESSER IMAGE.

Le paramètre type est une chaîne de 4 caractères indiquant le type de compression. Passez dans ce paramètre une des constantes du thème Compression images. Si type est une chaîne vide, image est chargée mais n'est pas compressée.

Le paramètre qualité est un Entier entre 1 et 1000 indiquant la qualité de l'image compressée. En général, réduire la qualité permet une meilleure compression de l'image.

Attention : Le ratio de compression possible pour une qualité donnée dépend de la taille de l'image et de l'image que vous voulez compresser. La compression de petites images peut ne pas produire de réduction de taille significative.

Exemple

L'exemple suivant affiche une boîte de dialogue d'ouverture de fichiers qui vous permet de sélectionner un fichier PICT. L'image est chargée en mémoire, compressée et stockée dans une variable de type Image, puis le fichier est refermé.

```
vRéf:=Ouvrir document ("";"PICT")
Si (OK=1)
    CHARGER ET COMPRESSER IMAGE(vRéf;QT Compresseur photo;500;vImage)
    FERMER DOCUMENT(vRéf)
Fin de si
```

Référence

COMPRESSER FICHIER IMAGE, COMPRESSER IMAGE, CONVERTIR IMAGE, ENREGISTRER IMAGE, Introduction aux images, LIRE FICHIER IMAGE.

COMBINER IMAGES

version 11

COMBINER IMAGES (imageRésultat; image1; opérateur; image2{; décalHoriz; décalVert})

Paramètre	Type		Description
imageRésultat	Image	←	Image résultant de la combinaison
image1	Image	→	Première image à combiner
opérateur	Entier long	→	Type de combinaison à effectuer
image2	Image	→	Seconde image à combiner
décalHoriz	Entier long	→	Décalage horizontal pour la superposition
décalVert	Entier long	→	Décalage vertical pour la superposition

Description

La commande COMBINER IMAGES permet de combiner les images image1 et image2 en mode opérateur pour en produire une troisième, imageRésultat. L'image résultat est de type composé et conserve toutes les caractéristiques des images sources.

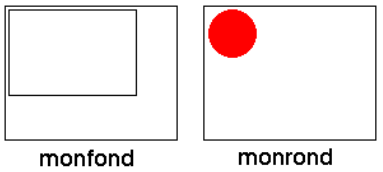
Note : Cette commande reprend et étend les fonctionnalités proposées par les opérateurs "classiques" de transformation d'images (+/, etc., cf. section Opérateurs sur les images). Ces opérateurs restent parfaitement utilisables dans 4D.

Passez dans opérateur le type de combinaison à appliquer. Trois types de combinaisons sont proposés, accessibles via des constantes placées dans le thème "Transformation des images" :
• Concaténation horizontale (1) : image2 est accolée à image1, le coin supérieur gauche de image2 coïncidant avec le coin supérieur droit de image1.
• Concaténation verticale (2) : image2 est accolée à image1, le coin supérieur gauche de image2 coïncidant avec le coin inférieur gauche de image1.
• Superposition (3) : image2 est placée par-dessus image1, le coin supérieur gauche de image2 coïncidant avec le coin supérieur gauche de image1.
Si les paramètres facultatifs décalHoriz et décalVert sont utilisés, une translation est appliquée à image2 avant la superposition. Les valeurs passées dans décalHoriz et décalVert doivent correspondre à des pixels. Passez des valeurs positives pour un décalage vers la droite ou vers le bas et une valeur négative pour un décalage vers la gauche ou vers le haut.

Note : La superposition effectuée par la commande COMBINER IMAGES diffère de la superposition proposée par les opérateurs "classiques" & et | (superposition exclusive et superposition inclusive). Tandis que la commande COMBINER IMAGES conserve les caractéristiques de chaque image source dans l'image résultante, les opérateurs & et | traitent chaque pixel et génèrent une image bitmap dans tous les cas. Ces opérateurs, conçus à l'origine pour les images monochromes, sont désormais obsolètes.

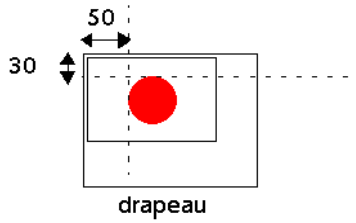
Exemple

Soient les images suivantes :



COMBINER IMAGES(drapeau;monfond;Superposition;monrond;50;30)

Résultat :



Référence

Opérateurs sur les images, TRANSFORMER IMAGE.

COMPRESSER FICHIER IMAGE

COMPRESSER FICHIER IMAGE (document; type; qualité)

Paramètre	Type		Description
document	RefDoc	→	Numéro de référence du document
type	Alpha	→	Type de compression
qualité	Numérique	→	Qualité de compression (1...1000)

Note de compatibilité : Cette commande fait appel à des mécanismes obsolètes et est conservée pour des raisons de compatibilité uniquement. Elle est avantageusement remplacée par les commandes ECRIRE FICHIER IMAGE ou IMAGE VERS BLOB.

Description

COMPRESSER FICHIER IMAGE compresse un document image sur disque. Utilisez cette commande pour compresser une image que vous savez ne pas pouvoir charger dans la mémoire disponible. Une fois compressée, elle peut être chargée en mémoire grâce à CHARGER ET COMPRESSER IMAGE.

Note : Cette commande ne fonctionne pas sous Windows.

Le paramètre type est une chaîne de 4 caractères indiquant le type de compression. Passez dans ce paramètre une des constantes du thème Compression images.

Le paramètre qualité est un entier entre 1 et 1000 indiquant la qualité de l'image compressée. En général, réduire la qualité permet une meilleure compression de l'image.

Attention : Le ratio de compression possible pour une qualité donnée dépend de la taille de l'image et de l'image que vous voulez compresser. La compression de petites images peut ne pas produire de réduction de taille significative.

Exemple

L'exemple suivant affiche une boîte de dialogue d'ouverture de fichiers qui vous permet de sélectionner un fichier PICT. Seuls les fichiers de type PICT seront affichés. L'image est compressée puis chargée en mémoire et stockée dans une variable de type Image. Le fichier est ensuite fermé.

```
vRéf:=Ouvrir document ("";"PICT")
Si (OK=1)
    COMPRESSER FICHIER IMAGE(vRef;QT Compresseur photo;500)
    CHARGER ET COMPRESSER IMAGE(vRéf;"";500;Image)
    FERMER DOCUMENT(vRéf)
Fin de si
```

Référence

CHARGER ET COMPRESSER IMAGE, COMPRESSER IMAGE, ENREGISTRER IMAGE.

COMPRESSER IMAGE

COMPRESSER IMAGE (image; type; qualité)

Paramètre	Type		Description
image	Image	→	Image à compresser
		←	Image compressée
type	Alpha	→	Type de compression (4 caractères)
qualité	Numérique	→	Qualité de compression (1...1000)

Note de compatibilité : Cette commande fait appel à des mécanismes obsolètes et est conservée pour des raisons de compatibilité uniquement. Elle est avantageusement remplacée par la commande CONVERTIR IMAGE.

Description

La commande COMPRESSER IMAGE compresse l'image contenue dans un champ ou une variable de type Image.

Le paramètre type est une chaîne de quatre caractères indiquant le type de compresseur. Passez dans ce paramètre une des constantes du thème Compression images.

Le paramètre qualité est un entier entre 1 et 1000 indiquant la qualité de l'image compressée. En général, réduire la qualité permet une meilleure compression de l'image.

Attention : Le ratio de compression possible pour une qualité donnée dépend de la taille de l'image et de l'image que vous voulez compresser. La compression de petites images peut ne pas produire de réduction de taille significative.

Référence

CHARGER ET COMPRESSER IMAGE, COMPRESSER FICHIER IMAGE, CONVERTIR IMAGE, Introduction aux images.

CONVERTIR IMAGE (image; codec)

Paramètre	Type		Description
image	Image	→	Image à convertir
		←	Image convertie
codec	Chaîne	→	Identifiant de codec d'image

Description

La commande CONVERTIR IMAGE convertit image dans un nouveau type.

Le paramètre codec indique le type d'image à générer. Un codec peut être une extension (par exemple ".gif"), un type Mime (par exemple "image/jpg") ou un code QuickTime sur 4 caractères (par exemple "PNTG"). Vous pouvez obtenir la liste des codecs disponibles via la commande LISTE CODECS IMAGES.

Si le champ ou la variable image est de type composé (si par exemple elle est issue d'un copier-coller), seules les informations correspondant au type codec sont conservées dans l'image résultante.

Exemple

Conversion de l'image vpPhoto au format jpeg :

```
CONVERTIR IMAGE(vpPhoto;".jpg")
```

Référence

LISTE CODECS IMAGES.

CREER IMAGETTE

version 11 (Modifiée)

CREER IMAGETTE (source; dest{; largeur{; hauteur{; mode{; profondeur}}}})

Paramètre	Type		Description
source	Image	→	Champ ou variable image 4D à passer en imagette
dest	Image	←	Imagette résultante
largeur	Entier	→	Largeur de l'imagette en pixels, Par défaut = 48
hauteur	Entier	→	Hauteur de l'imagette en pixels, Par défaut = 48
mode	Entier	→	Mode de création de l'imagette Par défaut = proportionnelle centrée (6)
profondeur	Entier	→	Obsolète, ne pas utiliser

Description

La commande CREER IMAGETTE retourne une imagette à partir d'une image source. Les imagettes sont généralement utilisées pour la prévisualisation d'images dans le cadre d'applications multimédia ou de sites Web.

Note : Cette commande ne nécessite pas la présence de QuickTime.

Passez dans source la variable ou le champ image 4D contenant l'image source à réduire sous forme d'imagette, et dans dest la variable ou le champ image 4D devant recevoir l'imagette résultante.

Les paramètres optionnels largeur et hauteur vous permettent de définir la taille en pixels de l'imagette que vous souhaitez obtenir. Si vous omettez ces paramètres, la taille par défaut de l'imagette sera de 48x48 pixels.

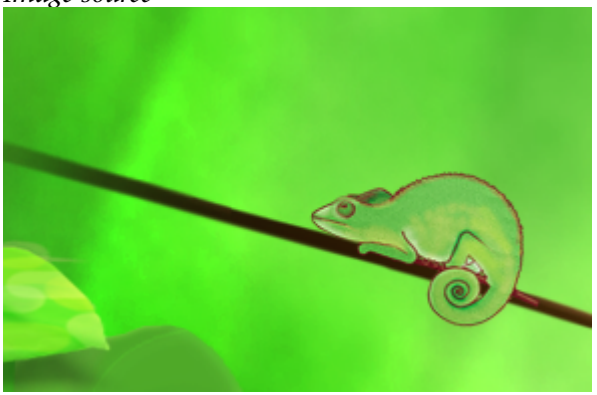
Le paramètre optionnel mode vous permet de définir le mode de création de l'imagette, c'est-à-dire la manière dont elle sera réduite. Vous pouvez utiliser l'un des trois modes suivants, accessibles par l'intermédiaire de constantes prédéfinies disponibles dans le thème "Formats d'affichage des images" :

Constante	Type	Valeur
Non tronquée	Entier long	2
Proportionnelle	Entier long	5
Proportionnelle centrée	Entier long	6 (défaut)

Note : Seules ces trois constantes peuvent être utilisées avec CREER IMAGETTE. Les autres constantes du thème "Formats d'affichage des images" ne s'appliquent pas à cette commande.

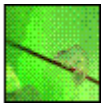
Si vous omettez le paramètre mode, le mode 6 (Proportionnelle centrée) est appliqué par défaut. Le résultat des différents modes est illustré ci-dessous :

Image source

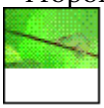


Imagettes résultantes (48x48)

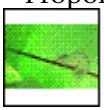
• Non tronquée = 2



• Proportionnelle = 5



• Proportionnelle centrée = 6 (mode par défaut)



Note : Avec les modes “Proportionnelle” et “Proportionnelle centrée”, les espaces vides apparaîtront blancs dans les imagettes — lorsque ces modes sont appliqués aux champs ou variables images dans les formulaires 4D, les espaces vides sont transparents.

A compter de la version 11 de 4D, le paramètre profondeur est ignoré et doit être omis. La commande utilise toujours la profondeur écran (nombre de couleurs) courante.

ECRIRE FICHIER IMAGE

version 11 (Modifiée)

ECRIRE FICHIER IMAGE (nomFichier; image{; codec})

Paramètre	Type		Description
nomFichier	Alpha	→	Nom ou chemin d'accès complet du fichier à écrire, ou chaîne vide
image	Image	→	Champ ou variable image à écrire
codec	Chaîne	→	Identifiant de codec d'image

Description

La commande ECRIRE FICHIER IMAGE vous permet de sauvegarder dans un fichier sur disque l'image passée dans le paramètre image, au format défini par codec.

Vous pouvez passer dans nomFichier le chemin d'accès complet du fichier à créer, ou uniquement le nom du fichier — auquel cas le fichier sera créé à côté du fichier de structure de la base. Sous Windows, vous devez également passer l'extension du fichier à créer.
Si vous passez une chaîne vide ("") dans nomFichier, la boîte de dialogue standard d'enregistrement de fichier apparaît, permettant à l'utilisateur de désigner le nom, l'emplacement et le format du fichier à créer.

Passez dans image la variable ou le champ image contenant l'image à stocker sur le disque.

Le paramètre optionnel codec vous permet de définir le format dans lequel l'image doit être sauvegardée. Un codec peut être une extension (par exemple ".gif"), un type Mime (par exemple "image/jpg") ou un code QuickTime sur 4 caractères (par exemple "PNTG"). Vous pouvez obtenir la liste des codecs disponibles via la commande LISTE CODECS IMAGES.
Si le paramètre codec est omis, le fichier image est créé au format PICT.

Si l'exécution de la commande est correcte, la variable système *Document* contient le chemin d'accès complet du fichier créé et la variable système *OK* prend la valeur 1. En cas d'échec, *OK* prend la valeur 0.

Référence

IMAGE VERS BLOB, Introduction aux images, LIRE FICHIER IMAGE, LISTE TYPES IMAGES.

ECRIRE IMAGE DANS BIBLIOTHEQUE

version 6.0.2

ECRIRE IMAGE DANS BIBLIOTHEQUE (image; refImage; nomImage)

Paramètre	Type		Description
image	Image	→	Nouvelle image
refImage	Numérique	→	Numéro de référence de l'image dans la bibliothèque d'images
nomImage	Alpha	→	Nouveau nom de l'image

Description

La commande ECRIRE IMAGE DANS BIBLIOTHEQUE crée une nouvelle image ou remplace une image existante dans la bibliothèque d'images.

Avant l'appel, vous passez :
• le numéro de référence de l'image dans refImage (compris entre 1 et 32767)
• l'image elle-même dans image.
• Le nom de l'image dans nomImage (longueur maximale : 255 caractères).

S'il existe déjà dans la bibliothèque une image possédant le même numéro de référence, son contenu est remplacé et elle est renommée avec les valeurs que vous avez passées dans image et nomImage.

Si aucune image ne possède le numéro de référence que vous avez passé dans refImage, une nouvelle image est créée dans la bibliothèque d'images.

4D Server : ECRIRE IMAGE DANS BIBLIOTHEQUE ne peut pas être utilisée dans une méthode exécutée sur le poste serveur (procédure stockée ou trigger). Si vous appelez ECRIRE IMAGE DANS BIBLIOTHEQUE sur le serveur, la commande ne fait rien, l'appel est ignoré.

Attention : Les objets de structure (éléments de listes hiérarchiques, lignes de menu, etc.) peuvent se référer à une image de la bibliothèque. Soyez prudent lorsque vous modifiez par programmation une image de la bibliothèque d'images.

Note : Si vous passez une image vide dans image, ou une valeur négative ou nulle dans refImage, la commande ne fait rien.

Exemples

(1) Quel que soit le contenu courant de la bibliothèque d'images, l'exemple suivant ajoute une nouvelle image dans la bibliothèque en cherchant d'abord un numéro de référence d'image unique :

```
LISTE IMAGES DANS BIBLIOTHEQUE($alRefImage;$asNomImage)
Repeter
    $vlRefImage:=1+Abs(Hasard)
Jusque (Chercher dans tableau($alRefImage;$vlRefImage)<0)
ECRIRE IMAGE DANS BIBLIOTHEQUE(vgImage;$vlRefImage;"Nouvelle Image")
```

(2) L'exemple suivant importe dans la bibliothèque des images stockées dans un document sur disque, créé par le troisième exemple de la commande LISTE IMAGES DANS BIBLIOTHEQUE :

```
REGLER SERIE(10;"")
Si (OK=1)
    RECEVOIR VARIABLE($vsTag)
    Si ($vsTag="4DV6BIBLIOTHEQUEIMAGEEXPORT")
        RECEVOIR VARIABLE($vlNbImages)
        Si ($vlNbImages>0)
            Boucle($vlImage;1;$vlNbImages)
                RECEVOIR VARIABLE($vlRefImage)
                Si (OK=1)
                    RECEVOIR VARIABLE($vsNomImage)
                Fin de si
                Si (OK=1)
                    RECEVOIR VARIABLE ($vgImage)
                Fin de si
                Si (OK=1)
                    ECRIRE IMAGE DANS BIBLIOTHEQUE($vgImage;$vlRefImage;
                                                                $vsNomImage)
                Sinon
                    $vlImage:=$vlNbImages+1
                    ALERTE("Ce fichier semble endommagé.")
                Fin de si
            Fin de boucle
        Sinon
            ALERTE("Ce fichier semble endommagé.")
        Fin de si
    Sinon
        ALERTE("Le fichier ""+Document+"" n'est pas un export de la bibliothèque
                                                                d'images.")
    Fin de si
    REGLER SERIE(11)
Fin de si
```

Référence

LIRE IMAGE DANS BIBLIOTHEQUE, LISTE IMAGES DANS BIBLIOTHEQUE, SUPPRIMER IMAGE DANS BIBLIOTHEQUE.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour retourner l'image, l'erreur –108 est générée. Notez que des erreurs d'E/S peuvent également être générées (si par exemple le fichier de structure est verrouillé). Vous pouvez intercepter ces erreurs avec une méthode de gestion d'erreurs.

ENREGISTRER IMAGE

ENREGISTRER IMAGE (document; image)

Paramètre	Type		Description
document	RefDoc	→	Numéro de référence du document
image	Image	→	Image à enregistrer

Note de compatibilité : Cette commande fait appel à des mécanismes obsolètes et est conservée pour des raisons de compatibilité uniquement. Elle est avantageusement remplacée par la commande ECRIRE FICHIER IMAGE.

Description

ENREGISTRER IMAGE enregistre image dans le document document, créé à l'aide de la fonction Creer document.

Exemple

L'exemple suivant crée un document et y stocke une image :

```
vRéf:=Creer document("";"PICT")
Si (OK=1)
    ENREGISTRER IMAGE(vRéf;vPict)
    FERMER DOCUMENT(vRéf)
Fin de si
```

Référence

ECRIRE FICHIER IMAGE.

IMAGE VERS BLOB

IMAGE VERS BLOB (image; blobImage; format)

Paramètre	Type		Description
image	Image	→	Champ ou variable image
blobImage	BLOB	←	BLOB devant contenir l'image convertie
format	Alpha (4)	→	Format d'image (4 caractères)

Description

La commande IMAGE VERS BLOB convertit une image stockée dans une variable ou un champ 4D dans un autre format, et place l'image résultante dans un BLOB.

Vous passez dans le paramètre image une variable ou un champ 4D de type image et dans le paramètre blobImage la variable ou le champ BLOB devant contenir l'image convertie.

Vous passez dans le paramètre format une chaîne de 4 caractères indiquant le format de conversion souhaité.
Ce format peut être :
• soit un code QuickTime (cf. description de la commande LISTE TYPES IMAGES, auquel cas QuickTime version 4 minimum devra être installé sur le poste,
• soit GIFf (format GIF) ou WBMP (Wireless Bitmap), ces deux codes ne nécessitant pas la présence de QuickTime.

Après l'exécution de la commande, blobImage contient l'image au format souhaité.

Si la conversion s'est déroulée correctement, la variable système *OK* prend la valeur 1. Si la conversion échoue (absence de QuickTime 4 ou convertisseur non disponible), *OK* prend la valeur 0 et le BLOB est généré vide (0 octet).

Référence

BLOB VERS IMAGE, ECRIRE FICHIER IMAGE, IMAGE VERS GIF, LISTE TYPES IMAGES.

IMAGE VERS GIF (imagePICT; blobGIF)

Paramètre	Type		Description
imagePICT	Image	→	Champ ou variable image
blobGIF	BLOB	←	BLOB contenant l'image de type GIF

Description

La commande IMAGE VERS GIF permet de créer une image au format GIF à partir d'une image (de type PICT) stockée dans une variable ou un champ 4D.

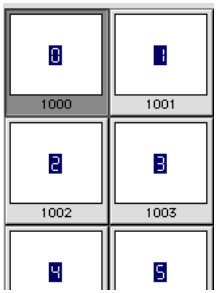
Vous passez dans le paramètre imagePICT une variable ou un champ 4D de type image, et dans le paramètre blobGIF, une variable ou un champ de type BLOB. Après l'exécution de la commande, blobGIF contient l'image au format GIF.

Note : Le format GIF est un format d'image comportant au plus 256 couleurs. Si l'image PICT d'origine en possède davantage, certaines couleurs seront perdues. La commande réduit le nombre de couleurs en fonction de la palette système. Le GIF généré est de type 87a (opaque) et normal (non entrelacé).

L'image incluse dans blobGIF pourra par la suite être enregistrée dans un fichier à l'aide de la commande BLOB VERS DOCUMENT ou être utilisée en vue d'une publication sur le Web. Si la conversion s'est déroulée correctement, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

Exemple

Vous souhaitez générer à la volée une image GIF affichant un compteur de connexions. Dans la bibliothèque d'images de la base, placez tous les chiffres sous forme d'images :



Dans la Méthode base Sur connexion Web, vous pouvez écrire :

```
    `Méthode base Sur connexion Web
Si (Contexte Web)
    ...
Sinon
    C_BLOB ($blob)
    Au cas ou
        ...
    : ($1="/4dcgi/counter") `Génération du compteur GIF
            `Lorsque 4D détecte cet URL lors de l'envoi de la page statique
        $blob:=gifcounter (<>nbHits)    `Calcul de l'image gif
            `La variable <>nbHits contient le nombre de connexions
        ENVOYER BLOB HTML ($blob;"image/gif")
            `Insertion de l'image et envoi au navigateur
    ...
    Fin de cas
Fin de si
```

Voici la méthode *gifcounter* :

```
    `Méthode projet gifcounter
C_ENTIER LONG($1)
C_IMAGE($img)
C_BLOB($0)
Si ($1=0)
    $ndigits:=1
Sinon
    $ndigits:=1+Longueur(Chaine($1))
Fin de si
Si ($ndigits<5)
    $ndigits:=5
Fin de si
$div:=10^($ndigits-1)
Boucle ($i;1;$ndigits)
    $ref:=Ent($1/$div)%10
    LIRE IMAGE DANS BIBLIOTHEQUE($ref+1000;picture)
    $img:=$img+picture
    $div:=$div/10
Fin de boucle
IMAGE VERS GIF($img;$0)
```

Lors de l'envoi de la page sur le browser, 4D affiche alors une image GIF du type suivant :

Variables et ensembles système

Si la conversion s'est déroulée correctement, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

Taille image

Taille image (image) → Numérique

Paramètre	Type		Description
image	Image	→	Image pour laquelle vous voulez connaître la taille en octets
Résultat	Numérique	←	Taille en octets de l'image

Description

Taille image retourne la taille de l'image image en octets.

Référence

PROPRIETES IMAGE.

LIRE FICHIER IMAGE

version 11.2 (Modifiée)

LIRE FICHIER IMAGE (nomFichier; image{; *})

Paramètre	Type		Description
nomFichier	Alpha	→	Nom ou chemin d'accès complet du fichier à lire, ou chaîne vide
image	Image	←	Champ ou variable recevant l'image
*	*	→	Si passé = accepter tout type de fichier

Description

La commande LIRE FICHIER IMAGE vous permet d'ouvrir l'image stockée dans le fichier disque désigné par nomFichier et de la placer dans le champ ou la variable 4D image.

Vous pouvez passer dans nomFichier le chemin d'accès complet du fichier à lire, ou uniquement le nom du fichier — auquel cas il doit se trouver à côté du fichier de structure de la base. Sous Windows, vous devez également passer l'extension du fichier.
Si vous passez une chaîne vide ("") dans nomFichier, la boîte de dialogue standard d'ouverture de documents apparaît, permettant à l'utilisateur de sélectionner le fichier à lire, ainsi que les formats disponibles.

Vous pouvez obtenir la liste des formats disponibles à l'aide de la commande LISTE CODECS IMAGES.

Passez dans image la variable ou le champ image devant recevoir l'image lue.

Note : Le format interne de l'image est conservé au sein de la variable ou du champ 4D. Dans le cas de formats personnalisés utilisant QuickTime, il sera nécessaire de disposer de QuickTime pour afficher l'image dans 4D.

Si vous passez le paramètre facultatif *, la commande acceptera tout type de fichier. Ce principe permet de manipuler des images sans nécessairement disposer des codecs adéquats (cf. description de la commande BLOB VERS IMAGE).

Référence

BLOB VERS IMAGE, ECRIRE FICHIER IMAGE, Introduction aux images, LISTE CODECS IMAGES, LISTE TYPES IMAGES.

Variables et ensembles système

Si l'exécution de la commande est correcte, la variable système *Document* contient le chemin d'accès complet du fichier ouvert et la variable système *OK* prend la valeur 1. En cas d'échec, *OK* prend la valeur 0.

LIRE IMAGE DANS BIBLIOTHEQUE (refImage | nomImage; image)

Paramètre	Type		Description
refImage | nomImage	Num | Alpha	→	Numéro de référence ou Nom d'une image de la bibliothèque d'images
image	Variable image	←	Image de la bibliothèque d'images

Description

La commande LIRE IMAGE DANS BIBLIOTHEQUE retourne dans image l'image de la bibliothèque dont vous avez passé le numéro de référence dans refImage ou le nom dans nomImage.

S'il n'existe pas d'image de ce numéro ou de ce nom dans la bibliothèque d'images, LIRE IMAGE DANS BIBLIOTHEQUE ne modifie pas le paramètre image.

Exemples

(1) L'exemple suivant retourne dans la variable vgMonImage l'image dont la référence est stockée dans la variable locale $vlRefImage :

```
LIRE IMAGE DANS BIBLIOTHEQUE($vlRefImage;vgMonImage)
```

(2) L'exemple suivant retourne dans la variable $DDcom_Prot_MonImage l'image nommée "DDcom_Prot_Bouton1" stockée dans la Bibliothèque d'images :

```
LIRE IMAGE DANS BIBLIOTHEQUE("DDcom_Prot_Bouton1";$DDcom_Prot_MonImage)
```

(3) Reportez-vous au troisième exemple de la commande LISTE IMAGES DANS BIBLIOTHEQUE.

Référence

ECRIRE IMAGE DANS BIBLIOTHEQUE, LISTE IMAGES DANS BIBLIOTHEQUE, SUPPRIMER IMAGE DANS BIBLIOTHEQUE.

Variables et ensembles système

La variable système OK prend la valeur 1 si l'image existe dans la bibliothèque d'images. Sinon, elle prend la valeur zéro.

Gestion des erreurs

S'il n'y a pas assez de mémoire pour retourner l'image, l'erreur –108 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs.

LISTE CODECS IMAGES

version 11

LISTE CODECS IMAGES (tabCodecs{; tabNoms})

Paramètre	Type		Description
tabCodecs	Tab Chaîne	←	Identifiants des codecs d'images disponibles
tabNoms	Tab Chaîne	←	Noms des codecs d'images

Description

La commande LISTE CODECS IMAGES remplit le tableau tabCodecs avec la liste des identifiants des codecs d'images disponibles sur la machine où elle est exécutée. Cette liste comporte à la fois les codecs des formats d'images gérés en natif par 4D ainsi que les identifiants des codecs QuickTime supplémentaires éventuellement installés sur le machine (cf. section Introduction aux images).
Ces identifiants peuvent être utilisés dans le paramètre format des commandes d'exportation d'images ECRIRE FICHIER IMAGE et IMAGE VERS BLOB.

Les identifiants des codecs peuvent être retournés dans le tableau tabCodecs sous trois formes :
• une extension (par exemple ".gif")
• un type Mime (par exemple "image/jpg")
• un code QuickTime sur 4 caractères (par exemple "PNTG")

La forme renvoyée par la commande dépend du mode de déclaration du codec au niveau du système d'exploitation. Le tableau facultatif tabNoms permet de récupérer le nom de chaque codec. Ces noms sont plus explicites que les identifiants. Ce tableau permet par exemple de construire et d'afficher un menu listant les codecs disponibles.

Référence

Introduction aux images, LISTE TYPES IMAGES.

LISTE IMAGES DANS BIBLIOTHEQUE (RefsImages; NomsImages)

Paramètre	Type		Description
RefsImages	Tab Entier long	←	Numéros de référence des images stockées dans la bibliothèque d'images
NomsImages	Tab Chaîne	←	Noms des images stockées dans la bibliothèque d'images

Description

La commande LISTE IMAGES DANS BIBLIOTHEQUE retourne les numéros de référence et le nom des images stockées dans la bibliothèque d'images de la base de données.

Après l'appel, vous récupérez les numéros de référence des images dans le tableau RefsImages et leurs noms dans le tableau NomsImages. Les deux tableaux sont synchronisés : le nième élément de RefsImages est le numéro de référence de l'image de la bibliothèque dont le nom est retourné dans le nième élément de NomsImages.

Si nécessaire, la commande crée et dimensionne automatiquement les tableaux RefsImages et NomsImages.
La longueur maximale du nom d'une image de la bibliothèque est de 255 caractères.

Si la bibliothèque d'images est vide, les deux tableaux retournés seront vides.

Pour obtenir le nombre d'images contenues dans la bibliothèque, il vous suffit de tester la taille d'un des deux tableaux à l'aide de la fonction Taille tableau.

Exemples

(1) Le code suivant retourne le contenu de la bibliothèque d'images dans les tableaux telRefImage et taNomImage :

```
LISTE IMAGES DANS BIBLIOTHEQUE(telRefImage;taNomImage)
```

(2) L'exemple suivant teste si la bibliothèque d'images est vide ou non :

```
LISTE IMAGES DANS BIBLIOTHEQUE(telRefImage;taNomImage)
Si (Taille tableau(telRefImage)=0)
   ALERTE("La bibliothèque d'images est vide.")
Sinon
   ALERTE("La bibliothèque d'images contient "+Chaine(Taille tableau(tlRefImage))+
                                                          " images.")
Fin de si
```

(3) L'exemple suivant exporte la Bibliothèque d'Images vers un document stocké sur disque :

```
LISTE IMAGES DANS BIBLIOTHEQUE($alRefImage;$asNomImage)
$vlNbImages:=Taille tableau($alRefImage)
Si ($vlNbImages>0)
    REGLER SERIE(12;"")
    Si (OK=1)
        $vsTag:="4DV6PICTURELIBRARYEXPORT"
        ENVOYER VARIABLE($vsTag)
        ENVOYER VARIABLE($vlNbImages)
        gError:=0
        Boucle($vlImage;1;$vlNbImages)
            $vlRefImage:=$alRefImage{$vlImage}
            $vsNomImage:=$asNomImage{$vlImage}
            LIRE IMAGE DANS BIBLIOTHEQUE($alRefImage{$vlImage};$vgImage)
            Si (OK=1)
                ENVOYER VARIABLE($vlRefImage)
                ENVOYER VARIABLE($vsNomImage)
                ENVOYER VARIABLE($vgImage)
            Sinon
                $vlImage:=$vlNbImages+1
                gError:=-108
            Fin de si
        Fin de boucle
        REGLER SERIE(11)
        Si (gError#0)
            ALERTE("La bibliothèque d'images n'a pas pu être exportée, recommencez
                                                    avec davantage de mémoire.")
            SUPPRIMER DOCUMENT (Document)
        Fin de si
    Fin de si
Sinon
    ALERTE("La bibliothèque d'images est vide.")
Fin de si
```

Référence

ECRIRE IMAGE DANS BIBLIOTHEQUE, LIRE IMAGE DANS BIBLIOTHEQUE, SUPPRIMER IMAGE DANS BIBLIOTHEQUE.

LISTE TYPES IMAGES (tabFormats{; tabNoms})

Paramètre	Type		Description
tabFormats	Tab Alpha (4)	←	Codes QuickTime des formats d'import/export disponibles
tabNoms	Tab Alpha	←	Noms des formats

Note de compatibilité : Cette commande est conservée pour des raisons de compatibilité. Toutefois, elle nécessite la présence de QuickTime et ne donne pas accès aux formats gérés en natif par 4D à compter de la version 11. Son intérêt est désormais limité et elle est avantageusement remplacée par la commande LISTE CODECS IMAGES.

Description

La commande LISTE TYPES IMAGES remplit le tableau tabFormats avec les codes QuickTime d'import/export d'images disponibles sur la machine où elle est exécutée.

Le tableau tabNoms, optionnel, permet de récupérer le nom de chaque format d'image. Les noms des formats sont plus explicites que leurs codes.

Bien entendu, cette commande requiert que QuickTime (version 4 minimum) soit installé sur la machine. Dans le cas contraire, tabFormats contient uniquement le format PICT.

LISTE TYPES IMAGES peut être utilisée pour vérifier la présence des formats d'images requis pour des bases. Cette fonction s'avère particulièrement utile lorsqu'une base emploie des formats personnalisés, non installés par défaut (possibilité offerte par QuickTime 4).
Les informations recueillies dans le tableau tabNoms permettent en outre de construire et d'afficher un pop up menu contenant les formats d'export d'images disponibles.

Codes de conversion QuickTime 4

Voici la liste des codes de conversion fournis en standard par QuickTime 4. Chaque code est formé impérativement de 4 caractères. A noter que cette liste peut varier en fonction des machines, QuickTime 4 permettant l'ajout de routines de conversion personnalisées.

Codes QuickTime 4	*Noms*
PICT	QuickDraw PICT (Mac OS)
PICS	PICS
GIFf	GIF (Graphic Interchange Format)
PNGf	PNG (Portable Netword Graphic)
TIFF	TIFF (Tagged Image File)
8BPS	Photoshop (2.5 et 3.0)
.SGI	Silicon Graphics

BMPf	BMP (Bitmap)
JPEG	JPEG (Joint Photographic Experts Group)
JPEG	JFIF
PNTG	MacPaint
TPIC	TGA (Targa)
qdgx	QuickDraw GX Picture (si QuickDraw GX installé)
qtif	QuickTime Image
FPix	FlashPix

Ces codes QuickTime d'import/export d'images sont utilisés en particulier par les commandes ECRIRE FICHIER IMAGE et LIRE FICHIER IMAGE.

Référence

BLOB VERS IMAGE, ECRIRE FICHIER IMAGE, IMAGE VERS BLOB, LIRE FICHIER IMAGE, LISTE CODECS IMAGES.

PROPRIETES IMAGE (image; largeur; hauteur{; hOffset{; vOffset{; mode}}})

Paramètre	Type		Description
image	Image	→	Image sur laquelle obtenir les informations
largeur	Numérique	←	Largeur de l'image exprimée en pixels
hauteur	Numérique	←	Hauteur de l'image exprimée en pixels
hOffset	Numérique	←	Offset horizontal lorsque l'image est affichée en arrière-plan
vOffset	Numérique	←	Offset vertical lorsque l'image est affichée en arrière-plan
mode	Numérique	←	Mode de transfert lorsque l'image est affichée en arrière-plan

Description

La commande PROPRIETES IMAGE retourne des informations sur l'image que vous avez passée dans le paramètre image.

Les paramètres largeur et hauteur reçoivent la largeur et hauteur réelles de l'image.

Les paramètres hOffset, vOffset et mode reçoivent la position et le mode de transfert de l'image lorsqu'elle est affichée en arrière-plan dans un formulaire (“Image sur fond”).

Référence

Taille image.

SUPPRIMER IMAGE DANS BIBLIOTHEQUE

version 6.7 (Modifiée)

SUPPRIMER IMAGE DANS BIBLIOTHEQUE (refImage | nomImage)

Paramètre	Type	Description
refImage | nomImage	Num | Alpha→	Numéro de référence ou Nom d'une image de la bibliothèque d'images

Description

La commande SUPPRIMER IMAGE DANS BIBLIOTHEQUE supprime de la bibliothèque d'images l'image dont vous avez passé le numéro de référence dans refImage ou le nom dans nomImage.
Si ce numéro de référence ou ce nom ne correspond à aucune image, la commande ne fait rien.

4D Server : SUPPRIMER IMAGE DANS BIBLIOTHEQUE ne peut pas être utilisée dans une méthode exécutée sur le poste serveur (procédure stockée ou trigger). Si vous appelez SUPPRIMER IMAGE DANS BIBLIOTHEQUE sur le serveur, il ne se passe rien, l'appel est ignoré.

Attention : Les objets de structure (éléments de liste hiérarchique, lignes de menu, etc.) peuvent se référer à une image de la bibliothèque. Soyez prudent lorsque vous supprimez par programmation une image de la bibliothèque d'images.

Exemples

(1) L'exemple suivant supprime l'image n°4444 de la bibliothèque d'images :

```
SUPPRIMER IMAGE DANS BIBLIOTHEQUE(4444)
```

(2) L'exemple suivant supprime de la bibliothèque d'images celles dont le nom commence par le symbole dollar ($) :

```
LISTE IMAGES DANS BIBLIOTHEQUE($alRefImage;$asNomImage)
Boucle($vlImage;1;Taille tableau($alRefImage))
    Si ($asNomImage{$vlImage}="$@")
        SUPPRIMER IMAGE DANS BIBLIOTHEQUE($alRefImage{$vlImage})
    Fin de si
Fin de boucle
```

Référence

ECRIRE IMAGE DANS BIBLIOTHEQUE, LIRE IMAGE DANS BIBLIOTHEQUE, LISTE IMAGES DANS BIBLIOTHEQUE.

TRANSFORMER IMAGE (image; opérateur{; param1{; param2{; param3{; param4}}}})

Paramètre	Type		Description
image	Image	→	Image source à transformer
		←	Image résultant de la transformation
opérateur	Entier long	→	Type de transformation à effectuer
param1	Numérique	→	Paramètre de la transformation
param2	Numérique	→	Paramètre de la transformation
param3	Numérique	→	Paramètre de la transformation
param4	Numérique	→	Paramètre de la transformation

Description

La commande TRANSFORMER IMAGE permet d'appliquer une transformation de type opérateur à l'image passée dans le paramètre image.

Note : Cette commande reprend et étend les fonctionnalités proposées par les opérateurs "classiques" de transformation d'images (+/, etc., cf. section Opérateurs sur les images). Ces opérateurs restent parfaitement utilisables dans 4D.

L'image source est modifiée directement à l'issue de l'exécution de la commande. A noter cependant qu'à l'exception de "Recadrage" et "Passage en niveaux de gris", les opérations ne sont pas destructives et permettent un retour en arrière via l'opération inverse ou l'opération "Réinitialisation". Par exemple, une image réduite à 1 % retrouvera sa taille originale sans altération si elle est agrandie 100 fois par la suite. Les transformations ne modifient pas le type d'origine de l'image : par exemple, une image vectorielle restera vectorielle à l'issue de la transformation.

Passez dans opérateur le numéro de l'opération à effectuer et dans param1 à param4 le ou les paramètre(s) nécessaire(s) à cette opération (le nombre de paramètres dépend de l'opération). Vous pouvez utiliser dans opérateur l'une des constantes du thème "Transformation des images". Ces opérateurs et leurs paramètres sont décrits dans le tableau suivant :

opérateur (valeur)	param1	param2	param3	param4	Valeurs
Réinitialisation (0)	-	-	-	-	
Redimensionnement (1)	Largeur	Hauteur	-	-	Facteurs
Translation (2)	Axe X	Axe Y	-	-	Pixels
Miroir horizontal (3)	-	-	-	-	
Miroir vertical (4)	-	-	-	-	
Recadrage (100)	Orig. X	Orig. Y	Largeur	Hauteur	Pixels
Passage en niveaux de gris (101)	-	-	-	-	

• Réinitialisation : toutes les opérations matricielles effectuées sur l'image (redimensionnement, miroir...) sont annulées.
• Redimensionnement : l'image est redimensionnée horizontalement et verticalement en fonction des valeurs passées respectivement dans param1 et param2. Ces valeurs sont des facteurs : par exemple, pour agrandir la largeur de 50 %, passez 1,5 dans param1 et pour réduire la hauteur de 50 %, passez 0,5 dans param2.
• Translation : l'image est déplacée de param1 pixels horizontalement et de param2 pixels verticalement. Passez une valeur positive pour un déplacement vers la droite ou vers le bas et une valeur négative pour un déplacement vers la gauche ou vers le haut.
• Miroir horizontal et Miroir vertical : l'effet miroir est appliqué à l'image d'origine. Tout déplacement éventuel effectué auparavant ne sera pas pris en compte.
• Recadrage : l'image est recadrée à partir du point de coordonnées param1 et param2 (exprimé en pixels). La largeur et la hauteur de la nouvelle image sont déterminées par les paramètres param3 et param4. Cette transformation ne peut pas être annulée.
• Passage en niveaux de gris : l'image est passée en niveaux de gris (aucun paramètre n'est requis). Cette transformation ne peut pas être annulée.

Exemple

Voici un exemple de recadrage (l'image est affichée dans le formulaire avec le format "Image tronquée (non centrée)") :

TRANSFORMER IMAGE($vpRouages;Recadrage;50;50;100;100)

Image d'origine | Image résultante

Référence

COMBINER IMAGES, Opérateurs sur les images.

30

Import-Export

ECRITURE DIF ({laTable; }document)

Paramètre	Type		Description
laTable	Table	→	Table de laquelle effectuer l'export ou Table par défaut si ce paramètre est omis
document	Alpha	→	Document DIF à exporter

Description

La commande ECRITURE DIF écrit dans document (document DIF Windows ou Mac OS) les données des enregistrements de la sélection courante de la table laTable du process courant.

L'opération d'export s'effectue par l'intermédiaire du formulaire sortie courant. L'opération d'export écrit les champs et les variables en fonction de l'ordre de saisie dans le formulaire sortie. C'est pourquoi vous devez veiller à utiliser un formulaire sortie ne contenant que les champs ou objets saisissables que vous voulez exporter. Ne placez pas de boutons ou autres objets sur le formulaire sortie. Les objets de sous-formulaire sont ignorés.

L'événement Sur chargement est envoyé à la méthode du formulaire pour chaque enregistrement exporté. Utilisez cet événement pour définir les variables utilisées dans le formulaire d'export.

Le paramètre document peut créer un nouveau document ou désigner un document existant. Si le document a le même nom qu'un document existant, ce dernier est remplacé. Le document peut contenir un chemin d'accès de volume et/ou de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard de sauvegarde de fichiers est affichée. Si l'utilisateur annule ce dialogue, l'opération d'export est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'export. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Si l'export s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande SUPPRIMER MESSAGES.

En mode Unicode (mode standard), la commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande UTILISER FILTRE pour modifier ce jeu de caractères.
En mode compatibilité ASCII, l'opération d'export utilise la table ASCII de la plate-forme de laquelle est effectué l'export, sauf si la commande UTILISER FILTRE a été préalablement utilisée.

Lors de l'utilisation de ECRITURE DIF, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrement est par défaut le retour chariot (code 13). Vous pouvez modifier ces valeurs en assignant de nouvelles valeurs aux deux variables système FldDelimit et RecDelimit. L'utilisateur peut modifier ces valeurs par défaut dans le dialogue d'export du mode Développement. Comme les champs Texte peuvent contenir des retours chariot, soyez prudent si vous utilisez le retour chariot comme délimiteur entre les champs à exporter.

Exemple

Cet exemple exporte des données vers un document DIF. Cette méthode commence par le choix du formulaire sortie, puis effectue l'export :

```
FORMULAIRE SORTIE([Personnes];"Export")
ECRITURE DIF([Personnes];"Nouvelles Personnes.dif")
    ` Export vers le document "Nouvelles Personnes.dif"
```

Référence

ECRITURE SYLK, EXPORTER TEXTE, LECTURE DIF, UTILISER FILTRE.

Variables et ensembles Système

OK prend la valeur 1 si l'export s'est correctement déroulé, sinon elle prend la valeur 0.

ECRITURE SYLK ({laTable; }document)

Paramètre	Type		Description
laTable	Table	→	Table de laquelle effectuer l'export ou Table par défaut si ce paramètres est omis
document	Alpha	→	Document SYLK à exporter

Description

La commande ECRITURE SYLK écrit dans document (document SYLK Windows ou Mac OS) les données des enregistrements de la sélection courante de la table laTable du process courant.

L'opération d'export s'effectue par l'intermédiaire du formulaire sortie courant. Les champs et les variables sont écrits en fonction de l'ordre de saisie dans le formulaire sortie. C'est pourquoi vous devez veiller à utiliser un formulaire sortie ne contenant que les champs ou objets saisissables que vous voulez exporter. Ne placez pas de boutons ou autres objets sur le formulaire sortie. Les objets de sous-formulaire sont ignorés.

L'événement Sur chargement est envoyé à la méthode du formulaire pour chaque enregistrement exporté. Utilisez cet événement pour définir les variables utilisées dans le formulaire d'export.

Le paramètre document peut créer un nouveau document ou désigner un document existant. Si le document a le même nom qu'un document existant, ce dernier est remplacé. Le document peut contenir un chemin d'accès de volume et/ou de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard de sauvegarde de fichiers est affichée. Si l'utilisateur annule ce dialogue, l'opération d'export est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'export. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Si l'export s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande SUPPRIMER MESSAGES.

En mode Unicode (mode standard), la commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande UTILISER FILTRE pour modifier ce jeu de caractères.
En mode compatibilité ASCII, l'opération d'export utilise la table ASCII de la plate-forme de laquelle est effectué l'export, sauf si la commande UTILISER FILTRE a été préalablement utilisée.

Lors de l'utilisation de ECRITURE SYLK, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrement est par défaut le retour chariot (code 13). Vous pouvez modifier ces valeurs en assignant de nouvelles valeurs aux deux variables système FldDelimit et RecDelimit. L'utilisateur peut modifier ces valeurs par défaut dans le dialogue d'export du mode Développement. Comme les champs Texte peuvent contenir des retours chariot, soyez prudent si vous utilisez le retour chariot comme délimiteur entre les champs à exporter.

Exemple

Cet exemple exporte des données vers un document SYLK. La méthode commence par le choix du formulaire sortie, puis l'export est déclenché :

```
FORMULAIRE SORTIE([Personnes];"Export")
ECRITURE SYLK([Personnes];"Nouvelles Personnes.slk")
    ` Export vers le document "Nouvelles Personnes.slk"
```

Référence

ECRITURE DIF, EXPORTER TEXTE, LECTURE SYLK, UTILISER FILTRE.

Variables et ensembles Système

OK prend la valeur 1 si l'export s'est correctement déroulé, sinon elle prend la valeur 0.

EXPORTER DONNEES (nomFichier{; projet{; *}})

Paramètre	Type		Description
nomFichier	Alpha	→	Chemin d'accès et nom du fichier d'export
projet	BLOB	→	Contenu du projet d'export
		←	Nouveau contenu du projet d'export (si le paramètre * a été passé)
*	*	→	Affichage de la boîte de dialogue d'export et mise à jour du projet

Description

La commande EXPORTER DONNEES permet d'exporter des données dans le fichier nomFichier. 4D peut exporter des données au format Texte, Texte de longueur fixe, XML, SYLK, DIF, DBF (dBase), et 4D.

Si vous passez une chaîne vide dans le nomFichier, EXPORTER DONNEES provoque l'affichage d'une boîte de dialogue standard d'enregistrement de fichiers, permettant à l'utilisateur de définir le nom, le type et l'emplacement du fichier d'export. Une fois la boîte de dialogue validée, la variable système *Document* contient le chemin d'accès et le nom de ce fichier. Si l'utilisateur clique sur le bouton **Annuler**, l'exécution est stoppée et la variable système *OK* prend la valeur 0.

• Si vous ne passez pas le paramètre optionnel projet, la boîte de dialogue de paramétrage d'export s'affiche alors. L'utilisateur peut définir ses paramètres d'export ou charger un projet d'export existant.

Note : Un projet d'export contient tous les paramètres de l'export, tels que les tables et champs exportés, les délimiteurs, etc. Vous définissez ces paramètres dans la boîte de dialogue d'export. Un projet peut être sauvegardé sur disque et chargé. Pour plus d'informations sur cette boîte de dialogue, reportez-vous au manuel *Mode Développement*.

• Si vous passez dans le paramètre projet un BLOB contenant un projet d'export valide, l'export s'effectue directement, sans intervention de l'utilisateur.
Dans ce cas, le projet doit avoir été préalablement défini dans la boîte de dialogue de paramétrage d'export, puis conservé. Pour cela, vous disposez de deux solutions :
- le sauvegarder sur disque, puis le charger à l'aide de la commande DOCUMENT VERS BLOB dans le champ ou la variable BLOB que vous passez dans le paramètre projet.
- utiliser la commande EXPORTER DONNEES avec un paramètre projet vide et le paramètre optionnel *, puis stocker le paramètre projet dans un champ BLOB (cf. ci-dessous). Cette solution permet, d'une part, de le conserver avec le fichier de données, et d'autre part d'éviter la phase de chargement dans un BLOB depuis le disque.

Le paramètre optionnel *, s'il est est spécifié, provoque l'affichage de la boîte de dialogue de paramétrage d'export avec les paramétrages définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini, tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, le paramètre projet contient, après la fermeture de la boîte de dialogue d'export, les paramètres du "nouveau" projet. Vous pouvez alors le stocker dans un champ BLOB, sur disque, etc.

Si l'export se déroule correctement, la variable système OK prend la valeur 1.

Exemple

Cet exemple crée un projet vide et y stockera les paramètres définis par l'utilisateur dans la boîte de dialogue d'export :

```
C_BLOB($exportParams)
FIXER TAILLE BLOB($exportParams;0)  `Initialisation du BLOB
EXPORTER DONNEES("DocExport.txt";$exportParams;*)
    ` Affichage de la boîte de dialogue d'export
```

Référence

ECRITURE DIF, ECRITURE SYLK, EXPORTER TEXTE, IMPORTER DONNEES.

Variables et ensembles système

Si l'utilisateur clique sur **Annuler** dans une des boîtes de dialogue (d'enregistrement de projet ou de paramétrage d'export), la variable système *OK* prend la valeur 0. Si l'export se déroule correctement, la variable système *OK* prend la valeur 1.

EXPORTER TEXTE ({laTable; }document)

Paramètre	Type		Description
laTable	Table	→	Table depuis laquelle effectuer l'export ou Table par défaut si ce paramètre est omis
document	Alpha	→	Document texte à exporter

Description

La commande EXPORTER TEXTE écrit dans document (document texte Windows ou Mac OS) les données des enregistrements de la sélection courante de la table laTable du process courant.

L'opération d'export s'effectue par le formulaire sortie courant. Les champs et les variables sont écrits en fonction de l'ordre de saisie dans le formulaire sortie. C'est pourquoi vous devez veiller à utiliser un formulaire sortie ne contenant que les champs ou objets saisissables que vous voulez exporter. Ne placez pas de boutons ou d'autres objets dans le formulaire sortie. Les objets de sous-formulaire sont ignorés.

L'événement Sur chargement est envoyé à la méthode du formulaire pour chaque enregistrement exporté. Utilisez cet événement pour définir les variables utilisées dans le formulaire d'export.

Le paramètre document peut créer un nouveau document ou désigner un document existant. Si le document a le même nom qu'un document existant, ce dernier est remplacé. Le document peut contenir un chemin d'accès de volume ou de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard de sauvegarde de fichiers est affichée. Si l'utilisateur annule ce dialogue, l'opération d'export est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'export. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Si l'export s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande SUPPRIMER MESSAGES.

En mode Unicode (mode standard), la commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande UTILISER FILTRE pour modifier ce jeu de caractères.
En mode compatibilité ASCII, l'opération d'export utilise la table ASCII de la plate-forme de laquelle est effectué l'export, sauf si la commande UTILISER FILTRE a été préalablement utilisée.

Lors de l'utilisation de EXPORTER TEXTE, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrement est par défaut le retour chariot (code 13). Vous pouvez modifier ces valeurs en assignant de nouvelles valeurs aux deux variables système FldDelimit et RecDelimit. L'utilisateur peut modifier ces valeurs par défaut dans le dialogue d'export du mode Développement. Comme les champs Texte peuvent contenir des retours chariot, soyez prudent si vous utilisez le retour chariot comme délimiteur entre les champs à exporter.

Exemple

Cet exemple exporte des données vers un document texte. Cette méthode commence par le choix du formulaire sortie. Ici, vous modifiez les délimiteurs et vous effectuez l'export :

```
FORMULAIRE SORTIE([Personnes];"Export")
FldDelimit:=27 ` caractère de délimitation : Escape
RecDelimit:=10 ` caractère de délimitation : Line Feed
EXPORTER TEXTE([Personnes];"Nouvelles Personnes.txt")
    ` Export vers le document "Nouvelles Personnes.txt"
```

Référence

ECRITURE DIF, ECRITURE SYLK, IMPORTER TEXTE, UTILISER FILTRE.

Variables et ensembles Système

OK prend la valeur 1 si l'export s'est correctement déroulé, sinon elle prend la valeur 0.

IMPORTER DONNEES (nomFichier{; projet{; *}})

Paramètre	Type		Description
nomFichier	Alpha	→	Chemin d'accès et nom du fichier à importer
projet	BLOB	→	Contenu du projet d'import
		←	Nouveau contenu du projet d'import (si le paramètre * a été passé)
*	*	→	Affichage de la boîte de dialogue d'import et mise à jour du projet

Description

La commande IMPORTER DONNEES permet d'importer des données depuis le fichier nomFichier. 4D peut importer des données au format Texte, Texte de longueur fixe, XML, SYLK, DIF, DBF (dBase), et 4D.

Si vous passez une chaîne vide dans le nomFichier, IMPORTER DONNEES provoque l'affichage d'une boîte de dialogue standard d'ouverture de fichiers, permettant à l'utilisateur de sélectionner le fichier d'import. Une fois la boîte de dialogue validée, la variable système *Document* contient le chemin d'accès et le nom du fichier d'import. Si l'utilisateur clique sur le bouton **Annuler**, l'exécution est stoppée et la variable système **OK** prend la valeur 0.

• Si vous ne passez pas le paramètre optionnel projet, la boîte de dialogue de paramétrage d'import s'affiche. L'utilisateur peut définir ses paramètres d'import ou charger un projet existant.

Note : Un projet d'import contient tous les paramètres de l'import, tels que les tables et champs d'arrivée, les délimiteurs, etc. Vous définissez ces paramètres dans la boîte de dialogue d'import. Un projet peut être sauvegardé sur disque et chargé. Pour plus d'informations sur cette boîte de dialogue, reportez-vous au manuel *Mode Développement*.

• Si vous passez dans le paramètre projet un BLOB contenant un projet d'import valide, l'import s'effectue directement, sans intervention de l'utilisateur.
Dans ce cas, le projet doit avoir été préalablement défini dans la boîte de dialogue de paramétrage d'import, puis conservé. Pour cela, vous disposez de deux solutions :
- le sauvegarder sur disque, puis le charger à l'aide de la commande DOCUMENT VERS BLOB dans le champ ou la variable BLOB que vous passez dans le paramètre projet.
- utiliser la commande IMPORTER DONNEES avec un paramètre projet vide et le paramètre optionnel *, puis stocker le paramètre projet dans un champ BLOB (cf. ci-dessous). Cette solution permet, d'une part, de le conserver avec le fichier de données, et d'autre part d'éviter la phase de chargement dans un BLOB depuis le disque.

Note : Reportez-vous à la commande EXPORTER DONNEES pour un exemple de définition de projet vide.

Le paramètre optionnel *****, s'il est est spécifié, provoque l'affichage de la boîte de dialogue de paramétrage d'import avec les paramétrages définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini, tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, le paramètre projet contient, après la fermeture de la boîte de dialogue d'import, les paramètres du "nouveau" projet. Vous pouvez alors le stocker dans un champ BLOB, sur disque, etc.

Si l'import se déroule correctement, la variable système *OK* prend la valeur 1.

Référence

EXPORTER DONNEES, IMPORTER TEXTE, LECTURE DIF, LECTURE SYLK.

Variables et ensembles système

Si l'utilisateur clique sur **Annuler** dans une des boîtes de dialogue (de sélection de projet ou de paramétrage d'import), la variable système *OK* prend la valeur 0. Si l'import se déroule correctement, la variable système *OK* prend la valeur 1.

IMPORTER TEXTE ({laTable; }document)

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle effectuer l'import ou Table par défaut si ce paramètre est omis
document	Alpha	→	Document texte à importer

Description

La commande IMPORTER TEXTE lit les données de document (document texte Windows ou Mac OS) et les écrit dans la table laTable en créant de nouveaux enregistrements.

L'opération d'import s'effectue par l'intermédiaire du formulaire entrée courant. Les champs et les variables sont lus en fonction du plan où ils se trouvent dans le formulaire entrée. Soyez donc attentif à ce qui se trouve au premier ou au deuxième plan en matière d'objets (champs et variables) dans le formulaire. Le premier objet dans lequel des données sont importées doit être à l'arrière-plan du formulaire. Si le nombre de champs et de variables dans le formulaire ne correspond pas au nombre de champs à importer, les champs supplémentaires sont ignorés. Un formulaire entrée utilisé pour l'import ne peut contenir de boutons. Les objets de sous-formulaire sont ignorés.

Note : Un moyen de s'assurer que les données sont importées dans les bons objets est de sélectionner l'objet dans lequel sera importé le premier champ. Cet objet doit être au premier plan. Pour l'import dans le deuxième champ, placer ce champ devant le premier et ainsi de suite, pour tous les champs et variables. Le champ qui se trouve au premier plan est donc celui dans lequel sont placées les dernières données du document à importer.

L'événement Sur validation est envoyé à la méthode du formulaire pour chaque enregistrement importé. Utilisez cet événement pour copier les données des variables vers les champs, si vous utilisez des variables dans le formulaire d'import.

Le paramètre document peut contenir un chemin d'accès aux noms de volumes et de dossiers. Si vous passez une chaîne vide, la boîte de dialogue standard d'ouverture de fichiers est affichée. Si l'utilisateur annule le dialogue, l'opération d'import est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'import. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Les enregistrements déjà importés le resteront. Si l'import s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande SUPPRIMER MESSAGES.

En mode Unicode (mode standard), la commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande UTILISER FILTRE pour modifier ce jeu de caractères.
En mode compatibilité ASCII, l'opération d'import utilise la table ASCII de la plate-forme de laquelle est effectué l'import, sauf si la commande UTILISER FILTRE a été préalablement utilisée.

Lors de l'utilisation de IMPORTER TEXTE, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrements par défaut est le retour chariot (code 13). Vous pouvez modifier ces valeurs par défaut en assignant de nouvelles valeurs aux variables système FldDelimit et RecDelimit. L'utilisateur peut modifier ces valeurs par défaut dans la boîte de dialogue d'import du mode Développement. Comme les champs Texte peuvent contenir des Retours chariot, soyez prudent si vous utilisez le Retour chariot comme délimiteur entre les champs à importer.

Exemple

Cet exemple importe des données d'un document texte. Cette méthode commence par le choix du formulaire. Ici, vous changez les délimiteurs, et vous faites l'import :

```
FORMULAIRE ENTREE([Personnes]; "Import")
FldDelimit:=27 ` caractère de délimitation : Escape
RecDelimit:=10 ` caractère de délimitation : Line Feed
IMPORTER TEXTE([Personnes];"Nouvelles Personnes.txt")
    ` Import du document “Nouvelles Personnes.txt”
```

Référence

EXPORTER TEXTE, LECTURE DIF, LECTURE SYLK, UTILISER FILTRE.

Variables et ensembles système

OK prend la valeur 1 si l'import s'est correctement déroulé, sinon elle prend la valeur 0.

LECTURE DIF ({laTable; }document)

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle effectuer l'import ou Table par défaut si ce paramètre est omis
document	Alpha	→	Document DIF à importer

Description

La commande LECTURE DIF lit les données de document (document DIF Windows ou Mac OS) et les écrit dans la table laTable en créant de nouveaux enregistrements.

L'opération d'import s'effectue par l'intermédiaire du formulaire entrée courant. Les champs et les variables sont lus en fonction du plan où ils se trouvent dans le formulaire entrée. Soyez donc attentif à ce qui se trouve au premier ou au deuxième plan en matière d'objets (champs et variables) dans le formulaire. Le premier objet dans lequel des données sont importées doit être à l'arrière-plan du formulaire. Si le nombre de champs et de variables dans le formulaire ne correspond pas au nombre de champs à importer, les champs supplémentaires sont ignorés. Un formulaire entrée utilisé pour l'import ne peut contenir de boutons. Les objets de sous-formulaire sont ignorés.

Note : Un moyen de s'assurer que les données sont importées dans les bons objets est de sélectionner l'objet dans lequel sera importé le premier champ. Cet objet doit être au premier plan. Pour l'import dans le deuxième champ, placer ce champ devant le premier et ainsi de suite, pour tous les champs et variables. Le champ qui se trouve au premier plan est donc celui dans lequel sont placées les dernières données du document à importer.

L'événement Sur validation est envoyé à la méthode du formulaire pour chaque enregistrement importé. Utilisez cet événement pour copier les données des variables vers les champs, si vous utilisez des variables dans le formulaire d'import.

Le paramètre document peut contenir un chemin d'accès aux noms de volumes et de dossiers. Si vous passez une chaine vide, la boîte de dialogue standard d'ouverture de fichiers est affichée. Si l'utilisateur annule le dialogue, l'opération d'import est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'import. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Les enregistrements déjà importés le resteront. Si l'import s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande SUPPRIMER MESSAGES.

En mode Unicode (mode standard), la commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande UTILISER FILTRE pour modifier ce jeu de caractères.
En mode compatibilité ASCII, l'opération d'import utilise la table ASCII de la plate-forme de laquelle est effectué l'import, sauf si la commande UTILISER FILTRE a été préalablement utilisée.

Lors de l'utilisation de LECTURE DIF, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrements par défaut est le retour chariot (code 13). Vous pouvez modifier ces valeurs par défaut en assignant de nouvelles valeurs aux variables système FldDelimit et RecDelimit. L'utilisateur peut modifier ces valeurs par défaut dans la boîte de dialogue d'import du mode Développement. Comme les champs Texte peuvent contenir des Retours chariot, soyez prudent si vous utilisez le Retour chariot comme délimiteur entre les champs à importer.

Exemple

Cet exemple importe des données d'un document DIF. Cette méthode commence par le choix du formulaire, puis effectue l'import :

```
FORMULAIRE ENTREE([Personnes]; "Import")
LECTURE DIF([Personnes];"Nouvelles Personnes.dif")
    ` Import du document “Nouvelles Personnes.dif”
```

Référence

ECRITURE DIF, IMPORTER TEXTE, LECTURE SYLK, UTILISER FILTRE.

Variables et ensembles Système

OK prend la valeur 1 si l'import s'est correctement déroulé, sinon elle prend la valeur 0.

LECTURE SYLK ({laTable; }document)

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle effectuer l'import ou Table par défaut si ce paramètre est omis
document	Alpha	→	Document SYLK à importer

Description

La commande LECTURE SYLK lit les données de document (document SYLK Windows ou Mac OS) et les écrit dans la table laTable en créant de nouveaux enregistrements.

L'opération d'import s'effectue par l'intermédiaire du formulaire entrée courant. Les champs et les variables sont lus en fonction du plan où ils se trouvent dans le formulaire entrée. Soyez donc attentif à ce qui se trouve au premier ou au deuxième plan en matière d'objets (champs et variables) dans le formulaire. Le premier objet dans lequel des données sont importées doit être à l'arrière-plan du formulaire. Si le nombre de champs et de variables dans le formulaire ne correspond pas au nombre de champs à importer, les champs supplémentaires sont ignorés. Un formulaire entrée utilisé pour l'import ne peut contenir de boutons. Les objets de sous-formulaire sont ignorés.

Note : Un moyen de s'assurer que les données sont importées dans les bons objets est de sélectionner l'objet dans lequel sera importé le premier champ. Cet objet doit être au premier plan. Pour l'import dans le deuxième champ, placer ce champ devant le premier et ainsi de suite, pour tous les champs et variables. Le champ qui se trouve au premier plan est donc celui dans lequel sont placées les dernières données du document à importer.

L'événement Sur validation est envoyé à la méthode du formulaire pour chaque enregistrement importé. Utilisez cet événement pour copier les données des variables vers les champs, si vous utilisez des variables dans le formulaire d'import.

Le paramètre document peut contenir un chemin d'accès aux noms de volumes et de dossiers. Si vous passez une chaine vide, la boîte de dialogue standard d'ouverture de fichiers est affichée. Si l'utilisateur annule le dialogue, l'opération d'import est annulée et la variable système OK est mise à 0.

Un thermomètre de progression est affiché pendant l'import. L'utilisateur peut annuler l'opération en cliquant sur le bouton **Stop**. Les enregistrements déjà importés le resteront. Si l'import s'est correctement déroulé, la variable système OK est mise à 1. En cas d'erreur ou d'interruption de l'opération, la variable système OK est mise à 0. Vous pouvez cacher le thermomètre au moyen de la commande SUPPRIMER MESSAGES.

En mode Unicode (mode standard), la commande utilise par défaut le jeu de caractères UTF-8. Vous pouvez utiliser la commande UTILISER FILTRE pour modifier ce jeu de caractères.
En mode compatibilité ASCII, l'opération d'import utilise la table ASCII de la plate-forme de laquelle est effectué l'import, sauf si la commande UTILISER FILTRE a été préalablement utilisée.

Lors de l'utilisation de LECTURE SYLK, le délimiteur de champs par défaut est le caractère de tabulation (code 9). Le délimiteur d'enregistrements par défaut est le retour chariot (code 13). Vous pouvez modifier ces valeurs par défaut en assignant de nouvelles valeurs aux variables système FldDelimit et RecDelimit. L'utilisateur peut modifier ces valeurs par défaut dans la boîte de dialogue d'import du mode Développement. Comme les champs Texte peuvent contenir des Retours chariot, soyez prudent si vous utilisez le Retour chariot comme délimiteur entre les champs à importer.

Exemple

Cet exemple importe des données d'un document SYLK. Cette méthode commence par le choix du formulaire puis déclenche l'import :

```
FORMULAIRE ENTREE([Personnes]; "Import")
LECTURE SYLK([Personnes];"Nouvelles Personnes.slk")
    ` Import du document “Nouvelles Personnes.slk”
```

Référence

ECRITURE SYLK, IMPORTER TEXTE, LECTURE DIF, UTILISER FILTRE.

Variables et ensembles Système

OK prend la valeur 1 si l'import s'est correctement déroulé, sinon elle prend la valeur 0.

31

Impressions

CUMULER SUR (objet{; objet2; ...; objetN})

Paramètre	Type	Description
objet	Champ | Var →	Champ ou variable de type numérique à cumuler

Description

CUMULER SUR désigne les champ(s) ou variable(s) à cumuler dans un état créé à l'aide de la commande IMPRIMER SELECTION.

Vous **devez** appeler NIVEAUX DE RUPTURES et CUMULER SUR avant la génération de chaque état dans lequel vous voulez utiliser des ruptures. Ces commandes activent le traitement des ruptures pour un état. Pour plus d'informations, reportez-vous à la description de la commande Sous total.

Utilisez CUMULER SUR si vous souhaitez calculer des sous-totaux pour des champs ou variables numériques dans un état. CUMULER SUR indique à 4D qu'il faut stocker les sous-totaux pour chaque élément spécifié dans objet. Les sous-totaux sont cumulés pour chaque niveau de rupture spécifié par la commande NIVEAUX DE RUPTURES.

Exécutez CUMULER SUR avant d'imprimer un état à l'aide de IMPRIMER SELECTION.

Utilisez la fonction Sous total dans la méthode formulaire ou une méthode objet pour retourner le sous-total d'un des objets spécifié dans objet.

Exemple

Reportez-vous à l'exemple de la commande NIVEAUX DE RUPTURES.

Référence

IMPRIMER SELECTION, NIVEAUX DE RUPTURES, Sous total, TRIER.

FERMER TACHE IMPRESSION

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande FERMER TACHE IMPRESSION permet de refermer la tâche d'impression préalablement ouverte par la commande OUVRIR TACHE IMPRESSION et d'envoyer à l'imprimante courante le document d'impression éventuellement construit.

Une fois cette commande exécutée, l'imprimante redevient disponible pour d'autres impressions.

Référence

OUVRIR TACHE IMPRESSION.

FIXER APERCU IMPRESSION (aperçu)

Paramètre	Type		Description
aperçu	Booléen	→	Impression à l'écran (Vrai) ou non (Faux)

Description

La commande FIXER APERCU IMPRESSION vous permet de sélectionner ou de désélectionner l'option d'**aperçu** dans la boîte de dialogue standard d'impression. Si vous passez Vrai dans écran, l'option "à l'écran" sera cochée. Si vous passez Faux, elle ne sera pas cochée. Ce paramétrage est local au process et n'affecte pas les autres process ou utilisateurs.

Exemple

L'exemple suivant sélectionne l'option **A l'écran** pour afficher le résultat d'une recherche de clients à l'écran, puis la désélectionne.

```
CHERCHER([Clients])
Si (OK=1)
   FIXER APERCU IMPRESSION (Vrai)
   IMPRIMER SELECTION ([Clients] ; *)
   FIXER APERCU IMPRESSION (Faux)
Fin de si
```

Référence

IMPRIMER ENREGISTREMENT, IMPRIMER SELECTION, PARAMETRES IMPRESSION.

FIXER IMPRIMANTE COURANTE (nomImpr)

Paramètre	Type		Description
nomImpr	Chaîne	→	Nom de l'imprimante à utiliser

Description

Note : Cette commande ne fonctionne pas sous Mac OS 9. Sous Windows, elle requiert au minimum Windows 2000.

La commande FIXER IMPRIMANTE COURANTE permet de désigner l'imprimante à utiliser pour les impressions avec l'application 4D courante.

Passez dans le paramètre nomImpr le nom de l'imprimante à sélectionner. Pour obtenir la liste des imprimantes disponibles, utilisez au préalable la commande LISTE IMPRIMANTES. Si vous passez une chaîne vide dans nomImpr, l'imprimante courante définie dans le système sera utilisée.

La commande FIXER IMPRIMANTE COURANTE doit être appelée avant FIXER OPTION IMPRESSION afin que les options disponibles correspondent à l'imprimante sélectionnée. En revanche, FIXER IMPRIMANTE COURANTE doit être appelée après UTILISER PARAMETRES IMPRESSION (le cas échéant), sinon le paramétrage d'imprimante n'est pas conservé.

Cette commande peut être utilisée avec les commandes IMPRIMER SELECTION, IMPRIMER ETIQUETTES, IMPRIMER ENREGISTREMENT, Imprimer ligne, QR ETAT et s'applique à toutes les impressions de 4D, y compris en mode Développement.
Les commandes d'impression doivent impérativement être appelées avec le paramètre > (le cas échéant) afin que le paramétrage défini soit conservé.

Référence

Lire imprimante courante, LISTE IMPRIMANTES.

Variables et ensembles système

Si la sélection d'imprimante est correctement effectuée, la variable système *OK* prend la valeur 1. Dans le cas contraire (par exemple l'imprimante désignée est introuvable), la variable système *OK* prend la valeur 0 et l'imprimante courante est inchangée.

FIXER MARGE IMPRESSION

FIXER MARGE IMPRESSION (gauche; haut; droite; bas)

Paramètre	Type	Description
gauche	Numérique →	Marge gauche
haut	Numérique →	Marge supérieure
droite	Numérique →	Marge droite
bas	Numérique →	Marge inférieure

Description

La commande FIXER MARGE IMPRESSION permet de fixer les valeurs des différentes marges d'impression lors de l'utilisation de la commande Imprimer ligne.

Vous pouvez passer dans les paramètres gauche, haut, droite et bas une de ces valeurs :
• 0 = utiliser les marges papier
• -1 = utiliser les marges imprimante
• valeur > 0 = marge en pixels (rappelons qu'1 pixel en 72 dpi représente approximativement 0,4 mm)

Les valeurs des paramètres droite et bas sont relatives à la droite et au bas du papier.

Par défaut, 4D base ses impressions sur les marges imprimante. Une fois la commande FIXER MARGE IMPRESSION exécutée, les paramètres modifiés seront conservés dans le même process pour toute la session.

Note : Pour plus d'informations sur la gestion des marges d'impression dans 4D, reportez-vous à la description de la commande LIRE MARGE IMPRESSION.

Exemples

(1) L'exemple suivant permet d'obtenir la taille de la marge morte :

```
FIXER MARGE IMPRESSION (-1;-1;-1;-1) `Fixe la marge imprimante
LIRE MARGE IMPRESSION($g;$h;$d;$b)
    `$g, $h, $d et $b correspondent aux marges mortes de la feuille
```

(2) L'exemple suivant permet d'obtenir la taille du papier :

```
FIXER MARGE IMPRESSION (0;0;0;0) `Fixe la marge papier
LIRE ZONE IMPRESSION($hauteur;$largeur)
    `Pour du A4 : $hauteur=842 ; $largeur=595 pixels
```

Référence

Imprimer ligne, Lire hauteur imprimee, LIRE MARGE IMPRESSION.

FIXER OPTION IMPRESSION (option; valeur1{; valeur2})

Paramètre	Type		Description
option	Entier long	→	Numéro d'option
valeur1	Entier long | Alpha	→	Valeur 1 de l'option
valeur2	Entier long | Alpha	→	Valeur 2 de l'option

Description

La commande FIXER OPTION IMPRESSION permet de modifier par programmation la valeur d'une option d'impression. Chaque option définie à l'aide de cette commande est appliquée dans toute la base et durant toute la session tant qu'aucune autre commande modifiant les paramètres d'impression (UTILISER PARAMETRES IMPRESSION, IMPRIMER SELECTION sans le paramètre >, etc.) n'est appelée.

Le paramètre option vous permet de désigner l'option à modifier. Vous pouvez passer une valeur ou une des constantes prédéfinies du thème "Options d'impression".
Passez dans les paramètres valeur1 et (facultativement) valeur2 la ou les nouvelle(s) valeur(s) de l'option spécifiée. Le nombre et la nature des valeurs à passer dépend du type d'option spécifiée.

Le tableau suivant liste les options et leurs valeurs possibles :

option (constante)	valeur1	valeur2
1 (Option papier)	Nom	-
	Largeur	Hauteur
2 (Option orientation)	1=Portrait, 2=Paysage	-
3 (Option échelle)	Nombre (%)	-
4 (Option nombre copies)	Nombre	-
5 (Option alimentation)	*Windows uniquement* :	
	Indice (numéro)	-
8 (Option couleur)	*Windows uniquement* :	
	1=N/B, 2=Couleur	-
9 (Option destination)	1=Imprimante,	-
	2=Fichier (PC)/PS (Mac),	Chemin accès
	3=PDF (Mac),	Chemin accès
	5=Ecran (Mac)	-
11 (Option recto verso)	*Windows uniquement* :	
	0=Recto (défaut)	-
	1=Recto-verso	Reliure : 0=Gauche (défaut), 1=Haut
12 (Option nom document à imprimer)	Nom du document à imprimer	-

13 (Option mode impression Mac)	0=mode PDF, 1=mode Postscript	-
14 (Option masquer progression impr)	0=Afficher (défaut), 1=Masquer	-

• Option papier (1) : la liste de tous les noms de papiers disponibles peut être obtenue via la commande VALEURS OPTION IMPRESSION.
Vous pouvez passer soit le nom du papier dans valeur1 (et dans ce cas omettre valeur2), soit la largeur du papier dans valeur1 et sa hauteur dans valeur2. La largeur et la hauteur doivent être exprimées en pixels écran.
• Option orientation (2) : vous pouvez passer soit 1 (Portrait), soit 2 (Paysage) dans valeur1.
• Option échelle (3) : passez un pourcentage dans valeur1. Attention, certaines imprimantes ne permettent pas de modifier l'échelle. Si vous passez une valeur invalide, la propriété est remise à 100% au moment de l'impression.
• Option nombre copies (4) : passez le nombre de copies à imprimer dans valeur1.
• Option alimentation (5) : passez un numéro correspondant à l'indice, dans le tableau des bacs retourné par la commande VALEURS OPTION IMPRESSION, du bac papier à utiliser.
Note : Cette option est utilisable sous Windows uniquement.
• Option couleur (8) : passez dans valeur1 un code indiquant le mode de prise en charge de la couleur : 1=Noir et blanc (monochrome), 2=Couleur.
Note : Cette option est utilisable sous Windows uniquement.
• Option destination (9) : passez dans valeur1 un code indiquant le type de destination de l'impression : 1=Imprimante, 2=Fichier (PC)/PS (Mac), 3=Fichier PDF (Mac OS uniquement), 5=Ecran (option du pilote Mac OS X).
Si valeur1 est différent de 1 ou de 5, passez dans valeur2 un chemin d'accès pour le document résultant. Ce chemin sera utilisé jusqu'à ce qu'un autre chemin soit spécifié. Si un fichier du même nom existe déjà à l'emplacement de destination, il est remplacé. Sous Windows uniquement : si vous passez une chaîne vide dans valeur2 ou omettez ce paramètre, une boîte de dialogue d'enregistrement de fichier apparaît au moment de l'impression.
• Option recto-verso (11) : vous pouvez passer soit 0 (Recto ou standard), soit 1 (Recto-verso) dans valeur1. Si valeur1 vaut 1, vous pouvez définir la reliure à appliquer à l'aide de valeur2 : 0=Reliure à gauche (valeur par défaut), 1=Reliure en haut.
Note : Cette option est utilisable sous Windows uniquement.
• Option nom document à imprimer (12) : passez dans valeur1 le nom du document d'impression devant apparaître dans la liste des documents du serveur d'impression. Pour utiliser ou rétablir le fonctionnement standard (utilisation du nom de la méthode dans le cas d'une méthode, nom de la table pour un enregistrement, etc.), passez une chaîne vide dans valeur1.
Attention : le nom défini par cette instruction sera utilisé pour tous les documents d'impression de la session tant qu'un nouveau nom ou une chaîne vide ne sera pas passé(e).
• Option mode impression Mac (13) : passez 0 pour fixer l'impression en mode PDF (valeur par défaut) et 1 pour "forcer" l'impression en mode Postscript dans valeur1. Cette option n'a pas d'effet sous Windows.
Note : Sous Mac OS X, les impressions sont effectuées par défaut en mode PDF. Or, le pilote d'impression PDF ne prend pas en charge les images PICT encapsulant des informations Postscript — ces images sont générées notamment par des logiciels de dessin vectoriel.

Pour résoudre ce problème, cette option permet de modifier le mode d'impression sous Mac OS X pour la session courante. Attention, l'impression en mode Postscript peut entraîner des effets de bords indésirables.
• Option masquer progression impr (14) : passez 1 dans valeur1 pour masquer toutes les fenêtres de progression d'impression et 0 pour rétablir leur affichage (fonctionnement par défaut). Cette option est particulièrement utile dans le cadre des impressions en PDF sous Mac OS X.
Note : Il existe déjà une option d'affichage Progression de l'impression accessible via la boîte de dialogue des préférences (page Application/Options). Toutefois, elle est globale à l'application et ne masque pas toutes les fenêtres sous Mac OS X.

Une fois fixée à l'aide de cette commande, une option d'impression sera conservée durant toute la session pour l'application 4D entière. Elle sera utilisée par les commandes IMPRIMER SELECTION, IMPRIMER ETIQUETTES, IMPRIMER ENREGISTREMENT, Imprimer ligne, QR ETAT et par toutes les impressions de 4D, y compris en mode Développement.

Notes :
• Il est impératif d'utiliser le paramètre optionnel > avec les commandes IMPRIMER SELECTION, IMPRIMER ETIQUETTES, IMPRIMER ENREGISTREMENT et SAUT DE PAGE afin de ne pas réinitialiser les options d'impression définies à l'aide de la commande FIXER OPTION IMPRESSION.
• La commande FIXER OPTION IMPRESSION fonctionne avec les imprimantes PostScript uniquement.

Référence

FIXER IMPRIMANTE COURANTE, LIRE OPTION IMPRESSION, VALEURS OPTION IMPRESSION.

Variables et ensembles système

La variable système *OK* prend la valeur 1 si la commande a été exécutée correctement, sinon elle prend la valeur 0.

Gestion des erreurs

Si la valeur passée pour une option est invalide ou si elle n'est pas disponible sur l'imprimante, la commande retourne une erreur (que vous pouvez intercepter à l'aide d'une méthode de gestion d'erreur installée par la commande APPELER SUR ERREUR) et la valeur courante de l'option est inchangée.

Constantes

Thème Options d'impression.

FIXER TAQUET IMPRESSION

version 2003 (Modifiée)

FIXER TAQUET IMPRESSION (numTaquet; position{; *})

Paramètre	Type		Description
numTaquet	Numérique	→	Numéro de taquet
position	Numérique	→	Nouvelle position du taquet
*	*	→	Si passé = déplacer les marqueurs suivants Si omis = ne pas déplacer les marqueurs suivants

Description

La commande FIXER TAQUET IMPRESSION permet de définir la position d'un taquet au moment de l'impression. Combinée aux commandes Lire taquet impression, DEPLACER OBJET ou Imprimer ligne, cette commande permet d'ajuster la taille des zones d'impression.

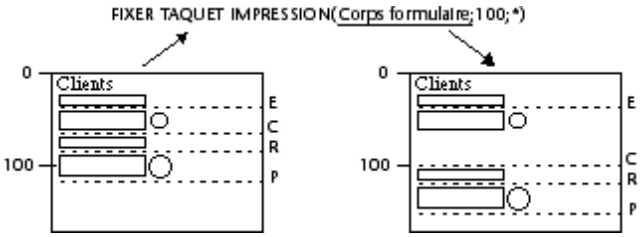
FIXER TAQUET IMPRESSION peut être appelée dans deux contextes :
• lors de l'événement formulaire Sur entête, dans le cadre de l'utilisation des commandes IMPRIMER SELECTION et IMPRIMER ENREGISTREMENT.
• lors de l'événement formulaire Sur impression corps, dans le cadre de l'utilisation de la commande Imprimer ligne. Ce fonctionnement facilite l'impression d'états personnalisés (voir exemple).
L'effet de la commande est limité à l'impression, aucune modification n'apparaît à l'écran. Les modifications apportées aux formulaires ne sont pas sauvegardées.

Passez dans le paramètre numTaquet une des constantes du thème "Zone de formulaire" :

Constante	Type	Valeur
Entête formulaire	Entier long	200
Entête formulaire1...10	Entier long	201...210
Corps formulaire	Entier long	0
Rupture formulaire0...9	Entier long	300...309
Pied de page formulaire	Entier long	100

Passez dans position la nouvelle position souhaitée du taquet, exprimée en pixels.

Si vous passez le paramètre optionnel *, tous les marqueurs situés au-dessous du marqueur désigné par numTaquet seront déplacés du même nombre de pixels et dans la même direction que lui lors de l'exécution de la commande. **Attention** : dans ce cas, les objets éventuellement présents dans les zones situées au-dessous du marqueur sont également déplacés.
Lorsque le paramètre * est utilisé, il est donc possible de positionner le marqueur numTaquet au-delà de la position initiale des marqueurs qui le suivent — ces derniers étant déplacés simultanément.



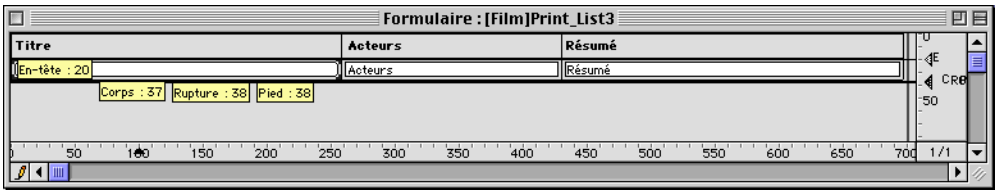
Notes :

• Cette commande modifie la position des taquets existants uniquement. Elle ne permet pas d'ajouter des taquets. Si vous désignez un taquet qui n'existe pas dans le formulaire, la commande ne fait rien.

• Le fonctionnement des taquets d'impression en mode Développement est conservé : un taquet ne peut pas aller plus haut que celui qui le précède ni plus bas que celui qui le suit (lorsque le paramètre * n'est pas utilisé).

Exemple

Cet exemple complet permet de générer l'impression d'un état sur trois colonnes, la hauteur de chaque ligne étant calculée à la volée en fonction du contenu des champs. Le formulaire de sortie utilisé pour l'impression est le suivant :



L'événement formulaire Sur impression corps a été sélectionné pour le formulaire (rappelons que la commande Imprimer ligne ne génère que cet événement, quelle que soit la zone imprimée).

Pour chaque enregistrement, la hauteur de la ligne doit être adaptée en fonction du contenu de la colonne "Acteurs" ou "Résumé" (colonne ayant le plus de contenu).

Voici le résultat souhaité :

Titre	Acteurs	Résumé
The Avengers	Ralph Fiennes, Uma Thurman, Sean Connery, Jim Broadbent, Patrick Macnee, Fiona Shaw, Eddie Izzard, Eileen Atkins	"The Avengers", the popular TV series from the sixties, is brought to the big screen with a mix of humour, retro fashion and action. Ralph Fiennes plays the role of well dressed John Steed and Uma Thurman is the beautiful Emma Peel dressed in a jumpsuit. Our two special agents fight crime in style. Sean Connery plays Sir August De Wynter, an evil genious that wants to take over the world with his high-tech weather machine. Unleashing snow storms or heatwaves, and armed with remote-controlled killer bees, this nut is a real menance to society. But all these climate changes won't stop our two heroes. A cup of tea, anyone?
20,000 Leagues Under The Sea		"The year is 1868, when New England's fishing harbors are the scene for a creature of unknown origin destroying ships at sea. It is the job of Professor Pierre Aronnax, a marine expert, and Ned Land, the iron-willed sailor, to learn the truth of the monster roaming the seas. The great novelist, Jules Verne, described this perilous journey to the darkest depths of the sea with Captain Nemo aboard the Nautilus."
The Adventures Of Ichabod And Mr. Toad (Walt Disney C	Bing Crosby, Basil Rathbone, Eric Blore, Pat O'Malley, John McLeish, Colin Campbell, Campbell Grant, Claud Allister	Hang on for the wild motorcar ride of J. Thaddeus Toad as he drives his friends Mole, Rat and Angus MacBadger into a worried frenzy! Then meet the spindly Ichabod Crane, who dreams of sweeping beautiful Katrina Van Tassel off her feet, despite opposition from town bully Brom Bones, who also has his eye on Katrina. The comic rivalry introduces Ichabod to the legend of the Headless Horseman resulting in a heart-thumping climax. Wonderfully narrated by Basil Rathbone and Bing Crosby, The Adventures Of Ichabod And Mr. Toad brims with high-spirited adventure, brilliant animation and captivating music you'll want to share with your family time and again.
Mission To Mars	Gary Sinise, Tim Robbins, Don Cheadle, Jerry O'Connell, Connie Nielsen	From the director of Mission Impossible comes the thrilling, eye-popping science-fiction adventure Mission To Mars – starring Gary Sinise (Snake Eyes) and Tim Robbins (Austin Powers: The Spy Who Shagged Me). The year is 2020, and the first manned mission to Mars, commanded by Luke Graham (Don Cheadle - Out Of Sight), lands safely on the red planet. But the Martian landscape harbors a bizarre and shocking secret that leads to a mysterious disaster so catastrophic, it decimates the crew. Haunted by a cryptic last message from Graham, NASA launches the Mars Recovery Mission to investigate and bring back survivors – if there are any. Confronted with nearly insurmountable dangers, but propelled by deep friendship, the team finally lands on Mars and makes a discovery so amazing, it takes your breath away. Mission To Mars is an action-packed rocket ride that will enthrall you with its stunning special effects and keep you on the edge of your seat.
The Abyss: Special Edition	Ed Harris, Mary Elizabeth Mastrantonio, Michael Biehn, Leo Burmeister, Todd Graff, John Bedford Lloyd, Kimberly Scott	In this thrilling, underwater action-adventure from writer-director James Cameron (Titanic, Terminator 2: Judgment Day, Aliens), a civilian oil-rig crew is recruited to conduct a search-and-rescue effort when a nuclear submarine mysteriously sinks. One diver (Ed Harris) soon finds himself on a spectacular odyssey over 25,000 feet below the ocean's surface, where he confronts a mysterious force that has the power to change the world or destroy it. Includes both the Special Edition, with 28 minutes of additional footage, and the original theatrical version, along with the 80-minute documentary Under Pressure: Making The Abyss, and much more.
Absence Of The Good	Stephen Baldwin, Rob Knepper, Shawn Huff, Allen Garfield, Silas Weir Mitchell, Tyne Daly	One cop. One killer. No clues. No dice. After his son is shot to death at school, Detective Caleb Barnes (Stephen Baldwin) loses touch with his soul. When a series of seemingly unrelated murders plagues Salt Lake City, the detective hides his grief in search for the killer. Hampered by a lack of clues and his commander's unrelenting pressure, Caleb painstakingly unravels a tangled web, exposing a malignant family history of abuse and murder.

La méthode projet d'impression est la suivante :

```
C_ENTIER LONG(vLhauteur_imp;$vLhauteur;vLhauteur_imprimee)
C_ALPHA(31;vSimpr_zone)
UTILISER PARAMETRES IMPRESSION([Film];"Print_List3")
LIRE ZONE IMPRESSION(vLhauteur_imp)
vLhauteur_imprimee:=0
TOUT SELECTIONNER([Film])
```

```
vSimpr_zone:="Entete"  `Impression de la zone d'en-tête
$vLhauteur:=Imprimer ligne([Film];"Print_List3";Entête formulaire )
$vLhauteur:=21  `Hauteur fixe
vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur

Tant que (Non(Fin de selection([Film])))
   vSimpr_zone:="Corps"  `Impression de la zone de corps
   $vLhauteur:=Imprimer ligne([Film];"Print_List3";Corps formulaire )
      `Le calcul du corps est effectué dans la méthode formulaire
   vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur
   Si (OK=0)  `NE PAS VALIDER a été exécutée dans la méthode formulaire
      SAUT DE PAGE
      vLhauteur_imprimee:=0
      vSimpr_zone:="Entete"  `Réimpression de la zone d'en-tête
      $vLhauteur:=Imprimer ligne([Film];"Print_List3";Entête formulaire )
      $vLhauteur:=21
      vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur
      vSimpr_zone:="Corps"
      $vLhauteur:=Imprimer ligne([Film];"Print_List3";Corps formulaire )
      vLhauteur_imprimee:=vLhauteur_imprimee+$vLhauteur
   Fin de si
   ENREGISTREMENT SUIVANT([Film])
Fin tant que
SAUT DE PAGE  `Assurons-nous que la dernière page est imprimée
```

La méthode du formulaire *Print_List3* est la suivante :

```
C_ENTIER LONG($g;$h;$d;$b;$larg_fix;$haut_préc;$g1;$h1;$d1;$b1)
C_ENTIER LONG($pos_finale;$i)
C_ENTIER LONG($position_c;$position_e;$hauteur_a_imprimer;$hauteur_restante)

Au cas ou
   : (vSimpr_zone="Corps")  `Impression du corps en cours
      LIRE RECT OBJET([Film]Acteurs;$g;$h;$d;$b)
      $larg_fix:=$d-$g  `Calcul de la taille du champ texte Acteurs
      $haut_préc:=$b-$h
      TAILLE OBJET OPTIMALE([Film]Acteurs;$larg;$haut;$larg_fix)
         `Taille optimale du champ en fonction du contenu
      $deplacement:=$haut-$haut_préc

      LIRE RECT OBJET([Film]Résumé;$g1;$h1;$d1;$b1)
      $larg_fix1:=$d1-$g1  `Calcul de la taille du champ texte Résumé
      $haut_préc1:=$b1-$h1
      TAILLE OBJET OPTIMALE([Film]Résumé;$larg1;$haut1;$larg_fix1)
                        `Taille optimale du champ en fonction du contenu
```

```
            $deplacement1:=$haut1-$haut_préc1
            Si ($deplacement1>$deplacement)
                  `On détermine le champ le plus haut
               $deplacement:=$deplacement1
            Fin de si

            Si ($deplacement>0)
               $position:=Lire taquet impression(Corps formulaire )
               $pos_finale:=$position+$deplacement
                  `On déplace le taquet Corps et ceux qui le suivent
               FIXER TAQUET IMPRESSION(Corps formulaire ;$pos_finale;*)
                  `Redimensionnement des zones de texte
               DEPLACER OBJET([Film]Acteurs;$g;$h;$d;$haut+$h;*)
               DEPLACER OBJET([Film]Résumé;$g1;$h1;$d1;$haut1+$h1;*)

                  `Redimensionnement des lignes de séparation
               LIRE RECT OBJET(*;"LigneH1";$g;$h;$d;$b)
               DEPLACER OBJET(*;"LigneH1";$g;$pos_finale-1;$d;$pos_finale;*)
               Boucle ($i;1;4;1)
                  LIRE RECT OBJET(*;"LigneV"+Chaine($i);$g;$h;$d;$b)
                  DEPLACER OBJET(*;"LigneV"+Chaine($i);$g;$h;$d;$pos_finale;*)
               Fin de boucle
            Fin de si

               `Calcul de la place disponible
            $position_c:=Lire taquet impression(Corps formulaire )
            $position_e:=Lire taquet impression(Entête formulaire )
            $hauteur_a_imprimer:=$position_c-$position_e
            $hauteur_restante:=hauteur_impression-vLhauteur_imprimee
            Si ($hauteur_restante<$hauteur_a_imprimer)   `Hauteur insuffisante
               NE PAS VALIDER   `Passer la ligne sur la page suivante
            Fin de si
      Fin de cas
```

Référence

DEPLACER OBJET, IMPRIMER ENREGISTREMENT, Imprimer ligne, IMPRIMER SELECTION, LIRE RECT OBJET, Lire taquet impression, SAUT DE PAGE, TAILLE OBJET OPTIMALE.

IMPRIMER ENREGISTREMENT ({table}{; }{* | >})

Paramètre	Type		Description
table	Table	→	Table de laquelle imprimer l'enregistrement courant ou Table par défaut si ce paramètre est omis
* | >	* | >	→	* pour supprimer les boîtes de dialogue d'impression ou > pour ne pas réinitialiser les paramètres d'impression

Description

Cette commande provoque l'impression de l'enregistrement courant de table, sans modifier la sélection courante. Le formulaire sortie courant est utilisé pour l'impression. S'il n'y a pas d'enregistrement courant dans table, IMPRIMER ENREGISTREMENT ne fait rien.

IMPRIMER ENREGISTREMENT permet d'imprimer des sous-formulaires, ce qui n'est pas possible avec la commande Imprimer ligne.

Note : Si l'enregistrement a subi des modifications qui n'ont pas encore été sauvegardées sur disque, la commande imprime les valeurs les plus récentes, et non celles stockées sur le disque.

Par défaut, IMPRIMER ENREGISTREMENT affiche les boîtes de dialogue d'impression. Si l'utilisateur annule une des deux boîtes de dialogue, l'exécution de la commande est stoppée.
Vous pouvez supprimer leur affichage en utilisant soit le paramètre optionnel astérisque (*), soit le paramètre optionnel "supérieur à" (>).
• Le paramètre * provoque une impression avec les paramètres d'impression courants (paramètres par défaut ou définis par les commandes UTILISER PARAMETRES IMPRESSION et/ou FIXER OPTION IMPRESSION).
• Le paramètre > provoque en outre l'impression sans réinitialisation des paramètres d'impression. Ce paramètre est utile lorsque vous souhaitez exécuter successivement plusieurs appels à IMPRIMER ENREGISTREMENT (par exemple à l'intérieur d'une boucle) tout en conservant des paramètres d'impression personnalisés préalablement définis.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :
• Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande avec le paramètre * ou >.
• En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Attention : N'utilisez pas la commande SAUT DE PAGE avec IMPRIMER ENREGISTREMENT. SAUT DE PAGE est exclusivement réservée à une utilisation combinée avec la commande Imprimer ligne.

Exemples

(1) L'exemple suivant imprime l'enregistrement courant de la table [Factures]. Cette méthode est celle d'un bouton **Imprimer** placé dans le formulaire entrée. Lorsque l'utilisateur clique sur ce bouton, l'enregistrement est imprimé dans un formulaire spécialement créé dans ce but.

```
    `Sélection du formulaire pour l'impression
FORMULAIRE SORTIE([Factures];"ImpressionEnregistrement")
    `Imprimer les factures (sans dialogues d'impression)
IMPRIMER ENREGISTREMENT([Factures];*)
FORMULAIRE SORTIE([Factures];"FormListe") `Restauration du formulaire sortie courant
```

(2) L'exemple suivant imprime le même enregistrement courant dans deux formulaires différents. Cette méthode est celle d'un bouton **Imprimer** placé dans un formulaire entrée. Vous souhaitez définir des paramètres d'impression personnalisés et les utiliser pour les deux formulaires.

```
PARAMETRES IMPRESSION   `L'utilisateur définit ses paramètres d'impression
Si (OK=1)
        `Utiliser un premier formulaire d'impression
    FORMULAIRE SORTIE([Employés];"Détaillé")
    IMPRIMER ENREGISTREMENT([Employés];>)
        `Imprimer en utilisant les paramètres définis par l'utilisateur
        `Utiliser un second formulaire d'impression
    FORMULAIRE SORTIE([Employés];"Simplifié")
    IMPRIMER ENREGISTREMENT([Employés];>)
        `Imprimer en utilisant les paramètres définis par l'utilisateur
    FORMULAIRE SORTIE([Employés];"Sortie")   `Rétablir le formulaire sortie par défaut
Fin de si
```

Référence

Imprimer ligne.

IMPRIMER ETIQUETTES ({table}{; étiqFichier{; * | >}})

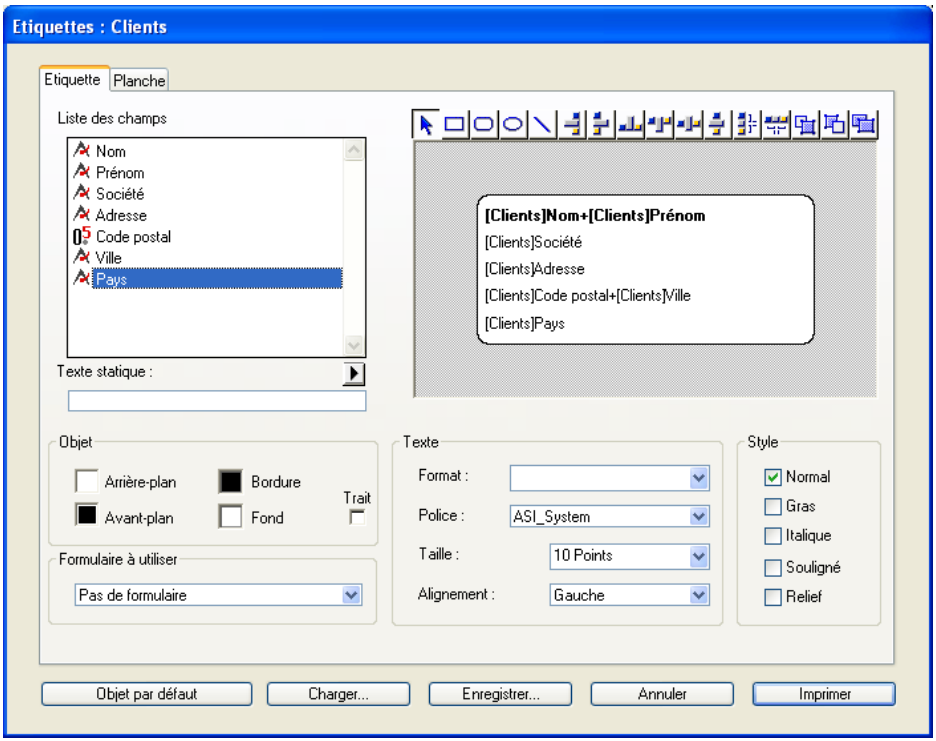
Paramètre	Type		Description
table	Table	→	Table à imprimer ou Table par défaut si ce paramètre est omis
étiqFichier	Alpha	→	Nom de fichier d'étiquettes sur disque
* | >	* | >	→	* pour supprimer les boîtes de dialogue d'impression ou > pour ne pas réinitialiser les paramètres d'impression

Description

IMPRIMER ETIQUETTES vous permet d'imprimer des étiquettes à partir des données de la sélection de table.

Si vous ne spécifiez pas le paramètre étiqFichier, IMPRIMER ETIQUETTES imprime la sélection courante de table sous forme d'étiquettes, en utilisant le formulaire sortie courant du process. Vous ne pouvez pas imprimer de sous-formulaires lorsque vous utilisez cette commande. Pour plus d'informations sur la création d'étiquettes à l'aide de formulaires, reportez-vous au manuel *Mode Développement* de 4D.

Si vous spécifiez le paramètre étiqFichier, IMPRIMER ETIQUETTES vous permet d'imprimer un document d'étiquettes existant stocké sur disque ou d'ouvrir l'Assistant de création d'étiquettes (affiché ci-dessous). Pour plus d'informations sur ce point, reportez-vous à l'exemple plus bas.



Par défaut, IMPRIMER ETIQUETTES affiche les boîtes de dialogue d'impression. Si l'utilisateur annule une des deux boîtes de dialogue, l'exécution de la commande est stoppée et l'état n'est pas imprimé.
Vous pouvez supprimer leur affichage en utilisant soit le paramètre optionnel astérisque (*), soit le paramètre optionnel “supérieur à” (>).
• Le paramètre * provoque une impression avec les paramètres d'impression courants (paramètres par défaut ou définis par les commandes UTILISER PARAMETRES IMPRESSION et/ou FIXER OPTION IMPRESSION).
• Le paramètre > provoque en outre l’impression sans réinitialisation des paramètres d’impression. Ce paramètre est utile lorsque vous souhaitez exécuter successivement plusieurs appels à IMPRIMER ETIQUETTES (par exemple à l'intérieur d’une boucle) tout en conservant des paramètres d’impression personnalisés préalablement définis.

Il est à noter que ces paramètres n'ont pas d'effet si l'assistant de création d'étiquettes est utilisé.

Si l'assistant de création d'étiquettes n'est pas utilisé, la variable système OK est mise à 1 si toutes les étiquettes ont été imprimées ; sinon, elle prend la valeur 0 (zéro) (par exemple si l'utilisateur a cliqué sur le bouton **Annuler** dans les boîtes de dialogue d'impression).

Si vous spécifiez le paramètre étiqFichier, les étiquettes sont imprimées avec les paramétrages définis dans étiqFichier. Si étiqFichier est une chaîne vide (""), IMPRIMER ETIQUETTES affiche une boîte de dialogue standard d'ouverture de documents, permettant à l'utilisateur de sélectionner le fichier d'étiquettes à utiliser. Si étiqFichier est le nom d'un fichier qui n'existe pas ou est invalide (si vous passez, par exemple, Caractere(1) dans étiqFichier), l'assistant de création d'étiquettes s'affiche, permettant à l'utilisateur de créer son propre format d'étiquettes.

Note : Si la table a été déclarée "invisible" en mode Développement, l'assistant de création d'étiquettes n'apparaît pas.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :
• Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande avec le paramètre * ou >.
• La syntaxe faisant apparaître l'éditeur d'étiquettes ne fonctionne pas avec 4D Server, dans ce cas, la variable système OK prend la valeur 0.
• En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemples

(1) L'exemple suivant imprime des étiquettes à l'aide du formulaire de sortie de la table. L'exemple s'appuie sur deux méthodes. La première est une méthode projet qui désigne le formulaire sortie à utiliser puis imprime les étiquettes :

```
TOUT SELECTIONNER([Adresses]) ` Sélection de tous les enregistrements
FORMULAIRE SORTIE ([Adresses]; "ImprimEtiq")` Sélection du formulaire sortie
IMPRIMER ETIQUETTES([Adresses]) ` Impression des étiquettes
FORMULAIRE SORTIE ([Adresses];"Sortie") ` Formulaire sortie par défaut
```

La seconde méthode est la méthode du formulaire "ImprimEtiq". Le formulaire contient une variable, nommée vEtiq, contenant les champs concaténés. Si le second champ Adresse (Adr2) est vide, il est enlevé par la méthode (notez que cette opération peut être effectuée automatiquement par l'assistant de création d'étiquettes). La méthode formulaire construit l'étiquette pour chaque enregistrement :

```
    ` Méthode formulaire [Adresses];"Etiquette sortie"
Au cas ou
    : (Evenement formulaire=Sur chargement)
        vEtiq:=[Adresses]Nom1+" "+[Adresses]Nom2+Caractere(13)+[Adresses]Adr1
                                                          +Caractere(13)
        Si ([Adresses]Adr2 # "")
            vEtiq:=vEtiq +[Adresses]Adr2+Caractere(13)
        Fin de si
        vEtiq:=vEtiq+[Adresses]Ville+", "+[Adresses]Département+" "+
                                                  [Adresses]Code Postal
                                                          Fin de cas
```

(2) L'exemple suivant permet à l'utilisateur d'effectuer une recherche sur la table [Employés], et d'imprimer automatiquement les étiquettes “Mes Etiquettes” :

```
CHERCHER ([Employés])
Si (OK=1)
    IMPRIMER ETIQUETTES ([Employés];"Mes Etiquettes";*)
Fin de si
```

(3) L'exemple suivant permet à l'utilisateur d'effectuer une recherche sur la table [Employés], puis de choisir les étiquettes qui doivent être imprimées :

```
CHERCHER ([Employés])
Si (OK=1)
    IMPRIMER ETIQUETTES ([Employés];"")
Fin de si
```

(4) L'exemple suivant permet à l'utilisateur d'effectuer une recherche sur la table [Employés], puis affiche l'assistant de création d'étiquettes afin que l'utilisateur puisse concevoir, sauvegarder, charger et imprimer tout type d'étiquettes :

```
CHERCHER ([Employés])
Si (OK=1)
    IMPRIMER ETIQUETTES ([Employés];Caractere(1))
Fin de si
```

Référence

IMPRIMER SELECTION, QR ETAT.

Imprimer ligne ({laTable; }formulaire{; zone1{; zone2}}){ → Numérique }

Paramètre	Type		Description
laTable	Table	→	Table à imprimer, ou Table par défaut si ce paramètre est omis
formulaire	Alpha	→	Formulaire à imprimer
zone1	Numérique	→	Marqueur d'impression, ou Zone de départ (si zone2 est spécifié)
zone2	Numérique	→	Zone de fin (si zone1 est spécifié)
Résultat	Numérique	←	Hauteur de la section imprimée

Description

La commande Imprimer ligne imprime simplement formulaire avec les valeurs courantes des champs et des variables de laTable. Cette commande est généralement utilisée pour imprimer des états particulièrement complexes nécessitant un contrôle total du processus d'impression. Imprimer ligne ne gère pas les traitements d'enregistrements, ni les ruptures, sauts de pages, en-têtes ou pieds de pages. Vous devez vous-même prendre en charge ces opérations. Imprimer ligne imprime uniquement des champs et des variables sous une forme fixe.

Comme la commande Imprimer ligne ne génère pas de saut de page après avoir imprimé un formulaire, elle vous permet de combiner facilement différents formulaires sur la même page. Ainsi, Imprimer ligne est idéale pour effectuer des impressions complexes impliquant plusieurs tables et plusieurs formulaires. Pour "forcer" 4D à effectuer un saut de page entre deux formulaires, utilisez la commande SAUT DE PAGE. Pour reporter sur la page suivante l'impression d'un formulaire dont la hauteur est supérieure à la place disponible, appelez la commande NE PAS VALIDER avant la commande SAUT DE PAGE.

Trois syntaxes peuvent être utilisées :

• **Impression du corps d'un formulaire**
Syntaxe :

```
hauteur:=Imprimer ligne (maTable;monFormulaire)
```

Dans ce cas, Imprimer ligne imprime uniquement la zone de Corps du formulaire (zone située entre les marqueurs En-tête et Corps).

• Impression de zone de formulaire
Syntaxe :

hauteur:=**Imprimer ligne** (maTable;monFormulaire;marqueur)

Dans ce cas, la commande imprimera la section désignée par marqueur. Passez dans le paramètre marqueur une des constantes du thème “Zone de formulaire” :

Constante	Type	Valeur
Entête formulaire	Entier long	200
Entête formulaire1...10	Entier long	201...210
Corps formulaire	Entier long	0
Rupture formulaire0...9	Entier long	300...309
Pied de page formulaire	Entier long	100

• Impression de section
Syntaxe :

hauteur:=**Imprimer ligne** (maTable;monFormulaire;zoneDébut;zoneFin)

Dans ce cas, la commande imprimera la section comprise entre les paramètres zoneDébut et zoneFin. Les valeurs saisies doivent être exprimées en pixels.

La valeur retournée par Imprimer ligne indique la hauteur de la zone d’impression. Cette valeur sera automatiquement prise en compte par la commande Lire hauteur imprimee.

Les boîtes de dialogue standard d'impression n'apparaissent pas lorsque vous utilisez la commande Imprimer ligne. L'état généré ne tient pas compte des paramètres d'impression définis en mode Développement pour le formulaire. Il y a deux manières de définir les paramètres d'impression avant d'effectuer une série d'appels à Imprimer ligne :
• Vous appelez PARAMETRES IMPRESSION. Dans ce cas, vous laissez l'utilisateur définir ses paramètres dans les boîtes de dialogue d'impression.
• Vous appelez UTILISER PARAMETRES IMPRESSION. Dans ce cas, les paramètres sont définis par programmation.

Imprimer ligne construit chaque page à imprimer en mémoire. La page n'est imprimée que lorsqu'elle est entièrement remplie ou lorsque vous appelez SAUT DE PAGE. Pour vous assurer que la dernière page d'une impression exécutée par l'intermédiaire Imprimer ligne soit effectivement imprimée, vous devez conclure par un appel à la commande SAUT DE PAGE. Sinon, la dernière page, si elle n'est pas pleine, reste en mémoire et n'est pas imprimée.

A compter de la version 2004.5 de 4D, cette commande imprime les zones et objets externes (par exemple des zones 4D Write ou 4D View). La zone est réinitialisée à chaque exécution de la commande.

Attention : Imprimer ligne n'imprime pas les sous-formulaires. Si vous voulez imprimer uniquement un formulaire comportant de tels objets, utilisez plutôt IMPRIMER ENREGISTREMENT.

Imprimer ligne ne génère qu'un événement Sur impression corps pour la méthode formulaire.

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :
• Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique).
• En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemples

(1) L'exemple suivant effectue la même chose que ce que ferait la commande IMPRIMER SELECTION. Cependant, l'état utilise deux formulaires différents suivant le type d'enregistrement (chèque émis ou dépôt) :

```
CHERCHER([Opérations]) ` Permettre à l'utilisateur de sélectionner les enregistrements
Si (OK=1)
   TRIER([Opérations]) ` Permettre à l'utilisateur de trier les enregistrements
      Si (OK=1)
         PARAMETRES IMPRESSION ` Afficher les boîtes de dialogue d'impression
         Si (OK=1)
            Boucle ($i; 1; Enregistrements trouves([Opérations]))
               Si ([Opérations]Type = "Chèque") ` Si c'est un chèque…
                  Imprimer ligne([Opérations]; "SortieChèque")
                     ` Utiliser un formulaire de chèque
               Sinon ` Sinon c'est un dépôt donc…
                  Imprimer ligne ([Opérations]; "SortieDépôt")
                     ` Utiliser un formulaire de dépôt...
               Fin de si
               ENREGISTREMENT SUIVANT([Opérations])
            Fin de boucle
            SAUT DE PAGE `S'assurer que la dernière page est imprimée
         Fin de si
      Fin de si
   Fin de si
```

(2) Reportez-vous à l'exemple de la commande FIXER TAQUET IMPRESSION.

Référence

NE PAS VALIDER, PARAMETRES IMPRESSION, SAUT DE PAGE, UTILISER PARAMETRES IMPRESSION.

IMPRIMER SELECTION ({table}{; }{* | >})

Paramètre	Type		Description
table	Table	→	Table à laquelle appartient la sélection à imprimer ou Table par défaut si ce paramètre est omis
* | >	* | >	→	* pour supprimer les boîtes de dialogue d'impression ou > pour ne pas réinitialiser les paramètres d'impression

Description

La commande IMPRIMER SELECTION imprime la sélection courante de table. Les enregistrements sont imprimés dans le formulaire sortie courant de la table du process en cours. IMPRIMER SELECTION a le même effet que la commande **Imprimer...** du mode Développement. Si la sélection courante est vide, IMPRIMER SELECTION ne fait rien.

Par défaut, IMPRIMER SELECTION affiche les boîtes de dialogue d'impression. Si l'utilisateur annule une des deux boîtes de dialogue, l'exécution de la commande est stoppée et l'état n'est pas imprimé.
Vous pouvez supprimer leur affichage en utilisant soit le paramètre optionnel astérisque (*), soit le paramètre optionnel “supérieur à” (>).
• Le paramètre * provoque une impression avec les paramètres d'impression courants (paramètres par défaut ou définis par les commandes UTILISER PARAMETRES IMPRESSION et/ou FIXER OPTION IMPRESSION).
• Le paramètre > provoque en outre l’impression sans réinitialisation des paramètres d’impression. Ce paramètre est utile lorsque vous souhaitez exécuter successivement plusieurs appels à IMPRIMER SELECTION (par exemple à l'intérieur d’une boucle) tout en conservant des paramètres d’impression personnalisés préalablement définis.

Pendant l'impression, la méthode du formulaire sortie et les méthodes objet du formulaire sont exécutées en fonction des événements sélectionnés dans les propriétés des formulaires et des objets, en mode Développement, ainsi que des événements effectivement générés :
• Un événement Sur entête est généré juste avant que la zone d'en-tête soit imprimée.
• Un événement Sur impression corps est généré juste avant que l'enregistrement soit imprimé.
• Un événement Sur impression sous total est généré juste avant qu'une zone de rupture soit imprimée.

• Un événement Sur impression pied de page est généré juste avant que la zone de pied de page soit imprimée.

Vous pouvez savoir si IMPRIMER SELECTION est sur le point d'imprimer le premier en-tête en testant Avant selection pendant un événement Sur entête. Vous pouvez également savoir si IMPRIMER SELECTION est sur le point d'imprimer le dernier pied de page, en testant Fin de selection pendant un événement Sur impression pied de page.
Pour plus d'informations, reportez-vous à la description de ces commandes ainsi qu'aux commandes Evenement formulaire et Niveau.

Si IMPRIMER SELECTION est appelée au même moment par deux process différents, l'impression déclenchée par le second process attendra que le premier ait terminé.

Pour imprimer une sélection triée avec des sous-totaux ou des ruptures à l'aide de la commande IMPRIMER SELECTION, vous devez d'abord trier la sélection. Puis vous devez inclure, dans chaque zone de rupture de l'état, une variable associée à une méthode objet assignant le sous-total à la variable. Vous pouvez aussi utiliser des fonctions statistiques ou arithmétiques telles que Somme et Moyenne pour assigner des valeurs aux variables. Pour plus d'informations, reportez-vous à la description des commandes Sous total, NIVEAUX DE RUPTURES et CUMULER SUR.

Attention : N'utilisez pas la commande SAUT DE PAGE avec IMPRIMER SELECTION. SAUT DE PAGE est exclusivement réservée à une utilisation combinée avec la commande Imprimer ligne.

Après un appel à IMPRIMER SELECTION, la variable OK prend la valeur 1 si l'impression s'est déroulée correctement. Si l'impression a été interrompue (par exemple l'utilisateur a cliqué sur un bouton Annuler dans les boîtes de dialogue d'impression), la variable OK prend la valeur 0 (zéro).

4D Server : Cette commande peut être exécutée sur 4D Server dans le cadre d'une procédure stockée. Dans ce contexte :
• Veillez à ce qu'aucune boîte de dialogue n'apparaisse sur le poste serveur (sauf besoin spécifique). Pour cela, il est nécessaire d'appeler la commande avec le paramètre * ou >.
• En cas de problème sur l'imprimante (plus de papier, imprimante déconnectée, etc.), aucun message d'erreur n'est généré.

Exemple

L'exemple suivant sélectionne la totalité des enregistrements de la table [Personnes]. La commande VISUALISER SELECTION est alors appelée pour afficher les enregistrements et permettre à l'utilisateur de sélectionner ceux qu'il souhaite imprimer. Enfin, les enregistrements choisis sont récupérés à l'aide de la commande UTILISER ENSEMBLE et imprimés par IMPRIMER SELECTION :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
VISUALISER SELECTION ([Personnes]; *) ` Affichage des enregistrements
UTILISER ENSEMBLE ("UserSet")
    ` Utiliser uniquement les enregistrements sélectionnés par l'utilisateur
IMPRIMER SELECTION ([Personnes]) ` Imprimer les enregistrements sélectionnés
```

Référence

CUMULER SUR, Niveau, NIVEAUX DE RUPTURES, Sous total, UTILISER PARAMETRES IMPRESSION.

Lire hauteur imprimee

Lire hauteur imprimee → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Position du marqueur

Description

La commande Lire hauteur imprimee retourne la hauteur globale (en pixels) de la section imprimée par la commande Imprimer ligne.

La valeur retournée sera comprise entre 0 (le bord supérieur de la page) et la hauteur globale retournée par la commande LIRE ZONE IMPRESSION (la taille maximum de la zone d'impression).

Si vous imprimez une nouvelle section via la commande Imprimer ligne, la hauteur de cette section est ajoutée à cette valeur. Si la zone d'impression disponible est insuffisante pour contenir cette section, une nouvelle page est générée et la valeur retournée est 0.

Les marges d'impression gauche et droite n'influent pas sur la valeur retournée, à la différence des marges inférieure et supérieure (définies éventuellement via la commande FIXER MARGE IMPRESSION).

Référence

FIXER MARGE IMPRESSION, Imprimer ligne, LIRE ZONE IMPRESSION.

Lire imprimante courante

Lire imprimante courante → Chaîne

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Chaîne	←	Nom de l'imprimante courante

Description

Note : Cette commande ne fonctionne pas sous Mac OS 9. Sous Windows, elle requiert au minimum Windows 2000.

La commande Lire imprimante courante retourne le nom de l'imprimante courante définie dans l'application 4D. Par défaut au lancement de 4D, l'imprimante courante est l'imprimante définie dans le système.

Si l'imprimante courante est gérée via un serveur d'impression ("spouleur"), le chemin d'accès complet (sous Windows) ou le nom du spouleur (sous Mac OS) est retourné.

Pour obtenir la liste des imprimantes disponibles ainsi que des informations complémentaires, utilisez la commande LISTE IMPRIMANTES. Pour modifier l'imprimante courante, utilisez la commande FIXER IMPRIMANTE COURANTE.

Référence

FIXER IMPRIMANTE COURANTE, LISTE IMPRIMANTES.

Variables et ensembles système

Si aucune imprimante n'est installée, la variable système *OK* prend la valeur 0. Sinon, *OK* prend la valeur 1.

LIRE MARGE IMPRESSION

LIRE MARGE IMPRESSION (gauche; haut; droite; bas)

Paramètre	Type		Description
gauche	Numérique	←	Marge gauche
haut	Numérique	←	Marge supérieure
droite	Numérique	←	Marge droite
bas	Numérique	←	Marge inférieure

Description

La commande LIRE MARGE IMPRESSION retourne les valeurs courantes des différentes marges définies lors de l'utilisation de la commande Imprimer ligne.

Les valeurs sont retournées en pixels par rapport au bord du papier.

Il est possible d'obtenir la taille du papier à l'aide de la fonction LIRE ZONE IMPRESSION, et ainsi de calculer la zone imprimable.

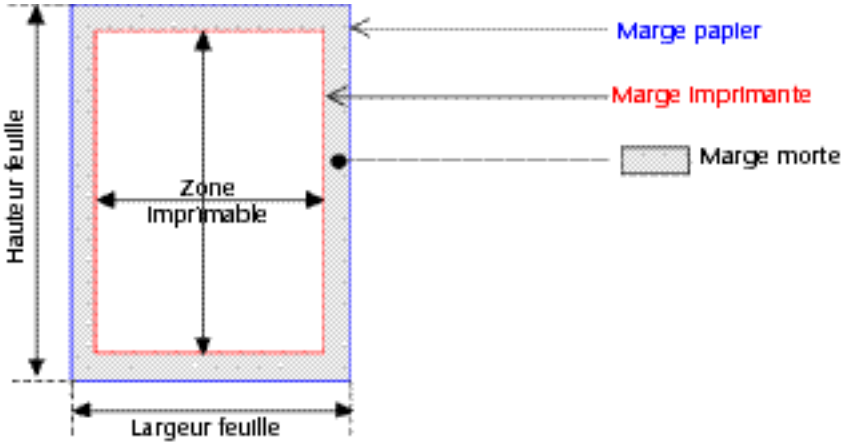
Gestion des marges d'impression

Par défaut, dans 4D le calcul des impressions est effectué sur la base des "marges imprimante". L'avantage de ce système est que les formulaires s'adaptent automatiquement aux nouvelles imprimantes (puisque positionnés dans la partie imprimable). En revanche, dans le cas des formulaires pré-imprimés, il n'était pas possible de positionner précisément les éléments à imprimer car un changement d'imprimante pouvait modifier les marges imprimante.

A compter de la version 6.8.1 de 4D, il est possible de baser l'impression des formulaires effectuée à l'aide des commandes Imprimer ligne, IMPRIMER ENREGISTREMENT et IMPRIMER SELECTION sur une marge fixe et identique sur chaque imprimante : la marge papier, c'est-à-dire les limites physiques de la feuille. Pour cela, il suffit d'utiliser les commandes LIRE MARGE IMPRESSION, FIXER MARGE IMPRESSION et LIRE ZONE IMPRESSION.

Terminologie des impressions

• **Marge papier** : la marge papier correspond aux limites physiques de la feuille.
• **Marge imprimante** : la marge imprimante est la marge au-dela de laquelle l'imprimante est incapable d'imprimer (pour des raisons physiques : galets d'impression, fin de course de la tête d'impression...). Elle varie d'une imprimante à l'autre et d'un format à l'autre.
• **Marge morte** : c'est la zone située entre la marge papier et la marge imprimante.



Référence

FIXER MARGE IMPRESSION, Imprimer ligne, LIRE ZONE IMPRESSION.

LIRE OPTION IMPRESSION (option; valeur1{; valeur2})

Paramètre	Type		Description
option	Entier long	→	Numéro d'option
valeur1	Entier long | Chaîne	←	Valeur 1 de l'option
valeur2	Entier long	←	Valeur 2 de l'option

Description

La commande LIRE OPTION IMPRESSION retourne la ou les valeur(s) courante(s) d'une option d'impression.

Le paramètre option vous permet de désigner l'option à lire. Vous pouvez passer une valeur ou une des constantes prédéfinies suivantes, placées dans le thème "Options d'impression" :

Constante	Type	Valeur
Option papier	Entier long	1
Option orientation	Entier long	2
Option échelle	Entier long	3
Option nombre copies	Entier long	4
Option alimentation	Entier long	5
Option couleur	Entier long	8
Option destination	Entier long	9
Option recto verso	Entier long	11
Option nom document à imprimer	Entier long	12
Option mode impression Mac	Entier long	13
Option masquer progression impr	Entier long	14

La commande retourne dans les paramètres valeur1 et (facultativement) valeur2 la ou les valeur(s) courante(s) de l'option spécifiée. Pour plus d'informations sur les options et les valeurs possibles, reportez-vous à la description de la commande FIXER OPTION IMPRESSION. A noter les spécificités suivantes de la commande LIRE OPTION IMPRESSION :
• option = 1 (Option papier) : retourne le nom du papier courant dans valeur1 si valeur2 est omis. Si valeur2 est passé, la commande retourne respectivement la largeur et la hauteur du papier dans valeur1 et valeur2. Utilisez la commande VALEURS OPTION IMPRESSION pour connaître le nom, la hauteur et la largeur de tous les formats de papier proposés par l'imprimante.
• option = 2 (Option orientation) : retourne 1 (Portrait) ou 2 (Paysage). Si une option d'orientation différente est utilisée, *valeur1* prend la valeur 0.

• option = 5 (Option alimentation) : retourne dans valeur1 l'indice, dans le tableau des bacs retourné par la commande VALEURS OPTION IMPRESSION, du bac papier utilisé (valeur2 doit être omis).
Note : Cette option est utilisable sous Windows uniquement.
• option = 8 (Option couleur) : retourne dans valeur1 un code indiquant le mode de prise en charge de la couleur : 1=Noir et blanc (monochrome), 2=Couleur.
Note : Cette option est utilisable sous Windows uniquement.
• option = 9 (Option destination) : si la valeur courante n'est pas dans la liste prédéfinie, valeur1 contient -1 et la variable système *OK* vaut 1. Si une erreur se produit, valeur1 et la variable système *OK* valent 0. Si valeur1 contient une valeur prédéfinie différente de 1 ou de 5, valeur2 contient le chemin d'accès du fichier imprimé.
• option = 11 (Option recto verso) : retourne 0 (Standard ou Recto, valeur par défaut) ou 1 (Recto-verso) dans valeur1. Si valeur1 vaut 1, valeur2 peut retourner une des valeurs suivantes : 0=Reliure à gauche (par défaut), 1=Reliure en haut.
Note : Cette option est utilisable sous Windows uniquement.
• option = 12 (Option nom document à imprimer) : retourne dans valeur1 le nom du document d'impression courant, s'il a été défini au préalable. Sinon, une chaîne vide est retournée.

Note : La commande LIRE OPTION IMPRESSION fonctionne avec les imprimantes PostScript uniquement.

Référence

FIXER OPTION IMPRESSION, VALEURS OPTION IMPRESSION.

Variables et ensembles système

La variable système *OK* prend la valeur 1 si la commande a été exécutée correctement, sinon elle prend la valeur 0.

Constantes

Thème Options d'impression.

Lire taquet impression

Lire taquet impression (numTaquet) → Numérique

Paramètre	Type		Description
numTaquet	Numérique	→	Numéro de taquet
Résultat	Numérique	←	Position du taquet

Description

La commande Lire taquet impression permet de récupérer la position courante d'un taquet lors d'une impression. Les coordonnées sont retournées en pixels (1 pixel = 1/72 pouce).

Cette commande peut être appelée dans deux contextes :
• lors de l'événement formulaire Sur entête, dans le cadre de l'utilisation des commandes IMPRIMER SELECTION et IMPRIMER ENREGISTREMENT.
• lors de l'événement formulaire Sur impression corps, dans le cadre de l'utilisation de la commande Imprimer ligne.

Passez dans le paramètre numTaquet une des constantes du thème "Zone de formulaire" :

Constante	Type	Valeur
Entête formulaire	Entier long	200
Entête formulaire1...10	Entier long	201...210
Corps formulaire	Entier long	0
Rupture formulaire0...9	Entier long	300...309
Pied de page formulaire	Entier long	100

Exemple

Reportez-vous à l'exemple de la commande FIXER TAQUET IMPRESSION.

Référence

DEPLACER OBJET, FIXER TAQUET IMPRESSION.

LIRE ZONE IMPRESSION (hauteur{; largeur})

Paramètre	Type		Description
hauteur	Numérique	←	Hauteur de la zone d'impression
largeur	Numérique	←	Largeur de la zone d'impression

Description

La commande LIRE ZONE IMPRESSION retourne dans les paramètres hauteur et largeur la taille en pixels de la zone d'impression. Cette taille dépend des paramètres d'impression courants, de l'orientation du papier, etc.

Les tailles retournées ne varient pas d'une page à l'autre (après un saut de page par exemple).

Associée à la commande Lire hauteur imprimee, cette commande est utile pour connaître le nombre de pixels disponibles pour l'impression, ou pour centrer un objet dans la page.

Pour connaître la taille totale de la page, vous pouvez :
• soit ajouter aux valeurs retournées par cette commande les marges fournies par la commande LIRE MARGE IMPRESSION.
• soit utiliser la syntaxe suivante :

```
FIXER MARGE IMPRESSION(0;0;0;0) ` Fixer la marge papier
LIRE ZONE IMPRESSION(hPapier;lPapier) ` Taille du papier
```

Note : Pour plus d'informations sur la gestion des marges d'impression dans 4D, reportez-vous à la description de la commande LIRE MARGE IMPRESSION.

Référence

Imprimer ligne, LIRE MARGE IMPRESSION.

LISTE IMPRIMANTES (tabNoms{; tabNomsAlt{; tabModèles}})

Paramètre	Type		Description
tabNoms	Tableau Texte	←	Noms des imprimantes
tabNomsAlt	Tableau Texte	←	Windows : Emplacements des imprimantes Mac OS : Noms personnalisés des imprimantes
tabModèles	Tableau Texte	←	Modèles des imprimantes (Windows uniquement)

Description

La commande LISTE IMPRIMANTES remplit le ou les tableau(x) passé(s) en paramètre(s) avec les noms ainsi que, facultativement, les emplacements ou les noms personnalisés et les modèles des imprimantes disponibles pour le poste.

Passez dans le paramètre tabNoms le nom d'un tableau texte. Après l'exécution de la commande, ce tableau contiendra la liste des noms d'imprimantes disponibles. Sous Mac OS, il s'agit des noms "système" fixes.

Note : Si les imprimantes sont gérées via un serveur d'impression ("spouleur"), le chemin d'accès complet (sous Windows) ou le nom du spouleur (sous Mac OS) est retourné.

Vous pouvez passer un deuxième tableau facultatif, tabNomsAlt. Le contenu de ce tableau dépend de la plate-forme :
• Sous Windows, vous récupérez pour chaque imprimante son emplacement réseau (ou son port local).
• Sous Mac OS, vous récupérez pour chaque imprimante son nom personnalisé, modifiable par l'utilisateur. Ce nom peut être utilisé par exemple dans des boîtes de dialogue.

Le paramètre facultatif tableModèles permet de récupérer le modèle de chaque imprimante. Ce paramètre est utilisable sous Windows uniquement.

Utilisez les commandes FIXER IMPRIMANTE COURANTE et Lire imprimante courante pour modifier ou connaître l'imprimante sélectionnée dans 4D. Vous devez leur passer les noms retournés dans le premier tableau (tabNoms).

Sous Windows, le nom d'une imprimante peut être modifié manuellement au niveau du système d'exploitation. En revanche, son emplacement et son modèle sont liés à ses caractéristiques physiques. Vous pouvez donc utiliser les valeurs des tableaux optionnels pour vérifier les caractéristiques de l'imprimante sélectionnée — typiquement, vous pouvez vérifier que tous les clients utilisent la même imprimante.
Sous Mac OS, cette vérification peut s'effectuer sur le nom de l'imprimante (nom du serveur d'impression), qui est le même pour chaque poste connecté.

Référence

FIXER IMPRIMANTE COURANTE, Lire imprimante courante.

Variables et ensembles système

La variable système *OK* prend la valeur 1 si la commande a été exécutée correctement, sinon elle prend la valeur 0 et les tableaux sont retournés vides.

Niveau → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Niveau de rupture ou d'en-tête courant

Description

La fonction Niveau sert à déterminer le niveau de rupture ou d'en-tête courant. Elle retourne le numéro du niveau de rupture pendant les événements Sur entête et Sur impression sous total.

Le niveau 0 est le dernier niveau à être imprimé et convient à l'impression d'un total général. Niveau retourne 1 lorsque 4D imprime une rupture sur le premier champ trié, 2 lorsque 4D imprime une rupture sur le deuxième champ trié, et ainsi de suite.

Exemple

Cet exemple est une maquette de méthode formulaire. Il traite chaque événement possible lorsqu'un état est imprimé dans un formulaire sortie. Niveau est appelé lorsqu'un en-tête ou une rupture est imprimé(e) :

```
    ` Méthode formulaire pour un formulaire sortie utilisé pour un état
$vpFormTable:=Table du formulaire courant
Au cas ou
        ` ...
    : (Evenement formulaire=Sur entête)
            ` La zone en-tête va être imprimée
        Au cas ou
            : (Avant selection($vpFormTable->))
                ` Le code pour la première rupture d'en-tête doit être placé ici
            : (Niveau = 1)
                ` Le code pour la rupture d'en-tête niveau 1 doit être placé ici
            : (Niveau = 2)
                ` Le code pour la rupture d'en-tête niveau 2 doit être placé ici
                ` ...
                                Fin de cas
```

```
: (Evenement formulaire=Sur impression corps)
      ` Un enregistrement va être imprimé
      ` Le code pour chaque enregistrement doit être placé ici
: (Evenement formulaire=Sur impression sous total)
      ` Une rupture va être imprimée
   Au cas ou
      : (Niveau = 0)
         ` Le code pour la rupture 0 doit être placé ici
      : (Niveau = 1)
         ` Le code pour la rupture 1 doit être placé ici
         ` ...
   Fin de cas
: (Evenement formulaire=Sur impression pied de page)
   Si (Fin de selection($vpFormTable->))
      ` Le code pour le dernier pied de page doit être placé ici
   Sinon
      ` Le code pour le pied de page doit être placé ici
   Fin de si
Fin de cas
```

Référence

CUMULER SUR, Evenement formulaire, IMPRIMER SELECTION, NIVEAUX DE RUPTURES.

NIVEAUX DE RUPTURES (niveau{; sautPage})

Paramètre	Type	Description
niveau	Numérique →	Nombre de niveaux de rupture
sautPage	Numérique →	Niveau de saut de page

Description

NIVEAUX DE RUPTURES spécifie le nombre de niveaux de rupture dans un état créé à l'aide de la commande IMPRIMER SELECTION.

Vous **devez** appeler NIVEAUX DE RUPTURES et CUMULER SUR avant la génération de chaque état dans lequel vous voulez utiliser des ruptures. Ces commandes activent le traitement des ruptures pour un état. Pour plus d'informations, reportez-vous à la description de la commande Sous total.

Le paramètre niveau indique le dernier niveau de rupture pour lequel vous voulez utiliser des ruptures. Ce nombre doit être inférieur ou égal aux niveaux de tris que vous aurez effectués avant l'impression. Si vous avez effectué un tri sur davantage de niveaux, ces niveaux seront imprimés triés, mais ne comporteront pas de rupture.

Chaque niveau de rupture généré provoquera l'impression de zones de rupture et d'en-tête dans le formulaire. Il doit y avoir au moins autant de zones de rupture dans le formulaire que la valeur que vous avez passée dans niveau. S'il y a davantage de zones de rupture, elles seront ignorées et ne seront pas imprimées.

Le second paramètre (optionnel), sautPage, permet de provoquer un saut de page sur le niveau de rupture de votre choix.

Exemple

L'exemple suivant imprime un état avec deux niveaux de rupture. La sélection est triée sur quatre champs, mais la commande NIVEAUX DE RUPTURES ne spécifie que deux niveaux de rupture. Seul un champ est cumulé à l'aide de la commande CUMULER SUR :

```
TRIER ([Emp]Service; >; [Emp]Titre; >; [Emp]Nom; >; Emp]Prénom; >)
   ` Trier sur quatre champs
NIVEAUX DE RUPTURES (2) ` Fixer 2 niveaux de rupture (Service et Titre)
CUMULER SUR ([Emp]Salaire) ` Cumuler sur les salaires
FORMULAIRE SORTIE ([Emp];"ServiceRessHum") ` Sélectionner le formulaire à imprimer
IMPRIMER SELECTION([Emp]) ` Imprimer l'état
```

Référence

CUMULER SUR, IMPRIMER SELECTION, Sous total, TRIER.

OUVRIR TACHE IMPRESSION

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande OUVRIR TACHE IMPRESSION ouvre une tâche d'impression (print job) et y empile tous les ordres d'impression exécutés par la suite, tant que la commande FERMER TACHE IMPRESSION n'est pas appelée. Cette commande vous permet de contrôler les tâches d'impression, et notamment de vous assurer qu'aucune tâche d'impression "parasite" ne puisse s'intercaler dans une séquence d'impressions.

La commande OUVRIR TACHE IMPRESSION peut être utilisée avec toutes les commandes d'impression de 4D, les commandes de l'éditeur d'états rapides ainsi que les commandes d'impression des plug-ins 4D Write et 4D View. En revanche, cette commande n'est pas compatible avec les plug-ins 4D Chart et 4D Draw, ainsi que la plupart des plug-ins tiers.

Lorsqu'une tâche est ouverte avec cette commande, l'imprimante est placée en mode "occupé" jusqu'à ce que l'impression soit effectivement lancée. Si un plug-in non compatible lance une impression dans ce contexte, l'erreur "imprimante occupée" est retournée.

Vous devez appeler la commande FERMER TACHE IMPRESSION pour terminer la tâche et envoyer le document d'impression à l'imprimante. Si vous omettez cette commande, le document d'impression restera dans la pile et l'imprimante sera inaccessible jusqu'à ce que vous quittiez l'application 4D.

La tâche d'impression est locale au process. Il est possible d'ouvrir autant de tâches d'impressions que de process. Bien entendu, dans ce cas il est nécessaire de disposer de plusieurs imprimantes car chaque imprimante est occupée jusqu'à la la fin de la tâche.

OUVRIR TACHE IMPRESSION utilise les paramètres d'impression courants (paramètres par défaut ou définis via les commandes UTILISER PARAMETRES IMPRESSION et/ou FIXER OPTION IMPRESSION). Les commandes modifiant les paramètres d'impression doivent être appelées avant OUVRIR TACHE IMPRESSION. Dans le cas contraire, une erreur est générée.

Référence

FERMER TACHE IMPRESSION.

Page impression → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Numéro de la page en cours d'impression

Description

Page impression retourne le numéro de la page en cours d'impression. Cette fonction vous permet de numéroter automatiquement les pages d'une impression en cours à l'aide de IMPRIMER SELECTION ou du menu Impression dans le mode Développement.

Exemples

L'exemple suivant change la position des numéros de page sur un état pour que l'état puisse être reproduit au format recto-verso. Le formulaire pour l'état comporte deux variables qui affichent les numéros de page. Une variable dans le coin bas à gauche (vNumGauche) imprime les numéros de page pairs. Une autre variable dans le coin bas à droite (vNumDroite) imprime les numéros de page impairs. L'exemple teste si le numéro de page est pair ou impair, puis utilise et efface les variables appropriées :

```
Au cas ou
   : (Evenement formulaire=Sur impression pied de page)
      Si ((Page impression % 2) = 0)  ` Modulo vaut 0 pour un numéro de page pair
         vNumGauche := Chaine (Page impression)
            ` Fixer le numéro de page à gauche
         vNumDroite := ""  ` Effacer le numéro de page droit
      Sinon  ` Sinon, le numéro de page est impair
         vNumGauche := ""  ` Effacer le numéro de page gauche
         vNumDroite := Chaine (Page impression)  ` Fixer le numéro de page à droite
      Fin de si
Fin de cas
```

Référence

IMPRIMER SELECTION.

PARAMETRES IMPRESSION

PARAMETRES IMPRESSION {(typeDial)}

Paramètre	Type		Description
typeDial	Entier long	→	Boîte(s) de dialogue à afficher : 0 (ou paramètre omis) = toutes, 1 = Format d'impression, 2 = Impression

Description

La commande PARAMETRES IMPRESSION provoque l'affichage d'une ou des deux boîtes de dialogue de paramétrage d'impression. Cette commande doit être appelée avant une série de commandes Imprimer ligne ou la commande OUVRIR TACHE IMPRESSION.

Le paramètre facultatif typeDial permet de configurer l'affichage des boîtes de dialogue d'impression :
• Si vous passez 0 dans typeDial ou omettez ce paramètre, les deux boîtes de dialogue d'impression sont affichées. La boîte de dialogue de paramétrage de l'impression s'affiche en premier lieu, suivie de celle d'impression proprement dite.
• Si vous passez 1 dans typeDial, seule la boîte de dialogue de paramétrage de l'impression est affichée. Les options d'impression courantes seront utilisées.
• Si vous passez 2 dans typeDial, seule la boîte de dialogue d'impression est affichée. Les paramètres d'impression courants seront utilisés.

Note : La boîte de dialogue d'impression comporte l'option **Imprimer à l'écran** permettant à l'utilisateur de visualiser son impression à l'écran. Vous pouvez présélectionner ou désélectionner cette option par un appel préalable à la commande FIXER APERCU IMPRESSION.

Exemple

Reportez-vous à l'exemple de la commande Imprimer ligne.

Variables et ensembles système

Si l'utilisateur clique sur le bouton OK dans chaque boîte de dialogue, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

Référence

FIXER APERCU IMPRESSION, Imprimer ligne, OUVRIR TACHE IMPRESSION, SAUT DE PAGE.

SAUT DE PAGE {(* | >)}

Paramètre	Type		Description
* | >		→	* Annule l'impression lancée par Imprimer ligne ou > Rend l'impression prioritaire

Description

La commande SAUT DE PAGE déclenche l'impression des données envoyées à l'imprimante et provoque un saut de page. SAUT DE PAGE s'utilise conjointement avec Imprimer ligne (dans le cadre de l'événement formulaire Sur impression corps) pour forcer des sauts de page et imprimer la dernière page créée en mémoire.
N'appelez pas SAUT DE PAGE avec la commande IMPRIMER SELECTION : dans ce cas, il est préférable d'utiliser les routines Sous total ou NIVEAUX DE RUPTURES avec leur paramètre optionnel pour générer des sauts de pages.

Les paramètres * et > sont optionnels.

Le paramètre * vous permet d'annuler une impression lancée avec la commande Imprimer ligne. L'exécution de cette instruction stoppe immédiatement l'impression en cours.

Note : Sous Windows, ce mécanisme peut être perturbé par les propriétés de "spouling" du serveur d'impression. Si l'imprimante est paramétrée de manière à lancer les impressions directement, l'annulation ne sera pas effective. Pour que l'instruction SAUT DE PAGE(*) fonctionne correctement, il est préférable d'affecter la propriété "Commencer l'impression une fois la dernière page spoulée" à l'imprimante.

Le paramètre > modifie le fonctionnement de la commande SAUT DE PAGE. Cette syntaxe provoque deux effets :

• Elle reporte l'impression jusqu'à ce que l'instruction SAUT DE PAGE sans paramètre soit de nouveau exécutée.

• Elle rend l'impression prioritaire. Aucune autre impression ne pourra s'intercaler tant que celle en cours ne sera pas terminée.
Cette seconde option est particulièrement intéressante lorsqu'elle est utilisée dans le cadre d'impressions en files d'attente. Le paramètre > garantit que l'impression sera réalisée à partir d'un seul fichier. Cela permet de réduire le temps d'impression.

Note : Lors d'une impression avec aperçu, si l'utilisateur clique sur le bouton d'annulation dans la boîte de dialogue de prévisualisation, la commande SAUT DE PAGE met la variable système OK à 0.

Exemples

(1) Reportez-vous à l'exemple de la commande Imprimer ligne.

(2) Reportez-vous à l'exemple de la commande FIXER TAQUET IMPRESSION.

Référence

Imprimer ligne, NE PAS VALIDER.

Sous total (valeurs{; sautPage}) → Numérique

Paramètre	Type		Description
valeurs	Champ	→	Champ ou variable numérique dont vous voulez calculer le sous-total
sautPage	Numérique	→	Niveau de rupture auquel effectuer un saut de page
Résultat	Numérique	←	Sous-total de valeurs

Description

Sous total retourne le sous-total de valeurs pour le niveau de rupture courant ou précédent. Sous total ne fonctionne que dans le cadre d'une sélection triée imprimée par l'intermédiaire de la commande IMPRIMER SELECTION ou de la commande de menu **Imprimer** du mode Développement. Le paramètre valeurs doit être de type numérique, entier ou entier long. Vous devez assigner le résultat de la fonction Sous total à une variable placée dans la zone de rupture du formulaire.

Attention : Vous **devez** utiliser les commandes NIVEAUX DE RUPTURES et CUMULER SUR avant d'imprimer un état sur lequel vous voulez traiter les niveaux de rupture et calculer des sous-totaux. Reportez-vous au paragraphe situé à la fin de cette section.

Le second paramètre (optionnel) de la fonction Sous total est utilisé pour provoquer des sauts de page lors de l'impression. Si sautPage vaut 0, Sous total ne génère aucun saut de page. Si sautPage vaut 1, Sous total génère un saut de page pour chaque niveau de rupture 1. Si sautPage vaut 2, Sous total génère un saut de page pour chaque niveau de rupture 1 et 2, etc.

Conseil : Si vous faites appel à la fonction Sous total dans le formulaire sortie affiché à l'écran, 4D va afficher un message d'erreur. La fermeture du dialogue d'erreur va provoquer un rafraîchissement de l'écran, donc de nouveau l'exécution de la méthode qui fait appel à Sous total, donc de nouveau un message d'erreur, etc. Pour sortir de ce cercle vicieux, appuyez sur les touche **Alt** + **Maj** (Windows) ou **Option**+**Maj** (Macintosh) et cliquez sur le bouton **Arrêter** dans la fenêtre d'erreur : cela met provisoirement fin aux rafraîchissements d'écran. Choisissez un autre formulaire de sortie pour éviter que le problème ne se répète. Passez en mode Structure pour isoler l'appel à la fonction Sous total par un test (Evenement formulaire = Sur impression sous total) si vous avez l'intention d'utiliser le même formulaire de sortie pour l'écran et l'imprimante.

Exemple

L'exemple suivant est la méthode objet d'une variable intitulée vSalaire, placée dans une zone de rupture d'un formulaire (R0, la zone située au-dessus du marqueur R0). La variable prend la valeur du sous-total du champ Salaire pour ce niveau de rupture. Le traitement des ruptures doit avoir été auparavant activé par les commandes CUMULER SUR et NIVEAUX DE RUPTURES.

```
Au cas ou
: (Evenement formulaire = Sur impression sous total)
    vSalaire := Sous total ([Employés]Salaire)
Fin de cas
```

Reportez-vous au chapitre "Les formulaires de sortie et les états" du manuel *Mode Développement* pour plus d'informations sur la construction de formulaires avec des niveaux de ruptures.

Traitement de niveaux de rupture dans les formulaires d'état

Pour pouvoir générer des états avec ruptures, vous devez déclencher le traitement des ruptures en appelant les commandes CUMULER SUR et NIVEAUX DE RUPTURES. Il faut que ces deux commandes soient appelées avant l'impression du formulaire. L'appel à la fonction Sous total est nécessaire pour afficher les calculs de niveaux intermédiaires. Il est obligatoire de trier sur au moins le nombre de niveaux de ruptures désiré.

Dans le cadre de l'utilisation des commandes CUMULER SUR et NIVEAUX DE RUPTURES, les étapes à suivre sont :
1. Sélectionner les enregistrements à imprimer,
2. Trier les enregistrements sur autant de niveaux que de niveaux de ruptures,
3. Appeler les commandes CUMULER SUR et NIVEAUX DE RUPTURES,
4. Imprimer l'état avec la commande IMPRIMER SELECTION.

La commande Sous total permet d'afficher des calculs de sous-totaux dans des formulaires.

Référence

CUMULER SUR, IMPRIMER SELECTION, Niveau, NIVEAUX DE RUPTURES.

UTILISER PARAMETRES IMPRESSION ({laTable; }formulaire)

Paramètre	Type		Description
laTable	Table	→	Table à laquelle appartient le formulaire ou Table par défaut si ce paramètre est omis
formulaire	Alpha	→	Formulaire à utiliser pour définir les paramètres d'impression

Description

UTILISER PARAMETRES IMPRESSION spécifie qu'une impression doit utiliser les paramètres mémorisés avec formulaire. Les paramètres d'impression sont stockés avec le formulaire au moment où il est sauvegardé en mode Développement.

Dans les trois cas suivants :
- un appel à IMPRIMER SELECTION auquel vous passez le paramètre optionnel *,
- un appel à IMPRIMER ENREGISTREMENT auquel vous passez le paramètre optionnel *,
- une série d'appels à Imprimer ligne non précédée d'un appel à PARAMETRES IMPRESSION,

... les boîtes de dialogue d'impression ne sont pas affichées et l'impression est effectuée avec les paramètres par défaut. Appeler UTILISER PARAMETRES IMPRESSION vous permet dans ce cas de ne pas afficher les boîtes de dialogue d'impression ET d'utiliser des paramètres d'impression qui ne sont pas ceux par défaut.

Exemple

Plusieurs formulaires (vides) sont créés pour une table nommée [Dessins]. Le format de page du formulaire “PS100” est fixé à 100%, le format de page du formulaire “PS90” est fixé à 90%, etc. La méthode projet suivante vous permet d'imprimer la sélection d'une table à différentes échelles sans devoir à chaque fois spécifier manuellement l'échelle dans les boîtes de dialogue d'impression (qui ne sont d'ailleurs pas affichées) :

```
    ` Méthode projet IMPRESSION ECHELLE AUTO
    ` IMPRESSION ECHELLE AUTO ( Pointeur ; Chaine {; Entier long } )
    ` IMPRESSION ECHELLE AUTO ( ->[Table]; "Form Sortie" {; Echelle } )
Si (Nombre de parametres>=3)
    UTILISER PARAMETRES IMPRESSION([Dessins];"PS"+Chaine($3))
    Si (Nombre de parametres>=2)
        FORMULAIRE SORTIE($1->;$2)
    Fin de si
Fin de si
Si (Nombre de parametres>=1)
    IMPRIMER SELECTION($1->;*)
Sinon
    IMPRIMER SELECTION(*)
Fin de si
```

Une fois que cette méthode projet est écrite, vous pouvez l'appeler ainsi :

```
    ` Recherche des factures courantes
CHERCHER ([Factures];[Factures]Payé=Faux)
    ` Impression d'un état réduit à 90%
IMPRESSION ECHELLE AUTO (->[Factures];"Etat Résumé";90)
    ` Impression d'un état réduit à 50%
IMPRESSION ECHELLE AUTO (->[Factures];"Etat détaillé";50)
```

Référence

FIXER OPTION IMPRESSION, IMPRIMER ENREGISTREMENT, Imprimer ligne, IMPRIMER SELECTION.

VALEURS OPTION IMPRESSION

VALEURS OPTION IMPRESSION (option; tabNoms{; tabInfo1{; tabInfo2}})

Paramètre	Type		Description
option	Entier long	→	Numéro d'option
tabNoms	Tableau Texte	←	Noms des valeurs
tabInfo1	Tableau Entier long	←	Valeurs 1 de l'option
tabInfo2	Tableau Entier long	←	Valeurs 2 de l'option

Description

La commande VALEURS OPTION IMPRESSION retourne dans le tableau tabNoms la liste des noms de valeurs disponibles pour l'option d'impression définie. Facultativement, vous pouvez récupérer des informations sur chaque valeur dans les tableaux tabInfo1 et tabInfo2.

Le paramètre option vous permet de désigner l'option à lire. Vous devez passer une des constantes du thème "Options d'impression" suivantes (options pouvant retourner des listes de noms de valeurs) :

Constante	Type	Valeur
Option papier	Entier long	1
Option alimentation	Entier long	5

Après exécution de la commande, le tableau tabNoms ainsi que, le cas échéant, les tableaux tabInfo1 et tabInfo2 seront remplis par la commande avec les noms et informations des valeurs disponibles.

Si vous passez la valeur 1 (Option papier) dans le paramètre option, la commande retournera les informations suivantes :
- dans le tableau tabNoms, les noms des formats de papiers disponibles ;
- dans le tableau tabInfo1, les hauteurs de chaque format de papier ;
- dans le tableau tabInfo2, les largeurs de chaque format de papier.

Note : Pour que vous puissiez obtenir ces informations, le pilote d'imprimante doit avoir accès à un fichier de description PostScript (PPD) valide de l'imprimante.

Pour utiliser un format de papier spécifique à l'aide de la commande FIXER OPTION IMPRESSION, vous pouvez passer soit une des valeurs du tableau tabNoms, soit les valeurs correspondantes des tableaux tabInfo1 et tabInfo2.

Si vous passez la valeur 5 (Option alimentation) dans le paramètre option, la commande retourne dans le tableau tabNoms les noms des différents bacs disponibles et leur numéro Windows interne dans tabInfo1 (tabInfo2 reste vide).
L'ordre des valeurs dans les tableaux est défini par le pilote d'impression. Pour désigner un bac à l'aide de la commande FIXER OPTION IMPRESSION, vous devez passer l'indice de l'élément souhaité dans tableau tabNoms ou tabInfo1.

Note : Cette option est utilisable sous Windows uniquement.

Pour plus d'informations sur les différentes options d'impression, reportez-vous à la description des commandes FIXER OPTION IMPRESSION et LIRE OPTION IMPRESSION.

Toutes les informations retournées par ces commandes sont fournies par le système d'exploitation. Reportez-vous à la documentation de votre système pour plus de détails sur certaines options.

Note : La commande VALEURS OPTION IMPRESSION fonctionne avec les imprimantes PostScript uniquement.

Référence

FIXER OPTION IMPRESSION, LIRE OPTION IMPRESSION.

Constantes

Thème Options d'impression.

32

Interface utilisateur

AFFICHER BARRE DE MENUS

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande AFFICHER BARRE DE MENUS rend visible la barre de menus.

Si la barre de menus était déjà visible, cette commande ne fait rien.

Exemple

Reportez-vous à l'exemple de la commande CACHER BARRE DE MENUS.

Référence

AFFICHER BARRE OUTILS, CACHER BARRE DE MENUS, CACHER BARRE OUTILS.

AFFICHER BARRE OUTILS

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande AFFICHER BARRE OUTILS rend visible la barre d'outils. Si la barre d'outils était déjà visible, AFFICHER BARRE OUTILS ne fait rien.

Référence

AFFICHER BARRE DE MENUS, CACHER BARRE DE MENUS, CACHER BARRE OUTILS.

APPELER SUR A PROPOS (libelléLigne; méthode)

Paramètre	Type		Description
libelléLigne	Alpha	→	Nouvelle ligne de menu A propos...
méthode	Alpha	→	Nom de la méthode à exécuter lorsque la ligne est choisie

Description

La commande APPELER SUR A PROPOS remplace la ligne de menu **A propos de 4D...** du menu **Aide** (sous Windows) ou du menu **application** (Mac OS X) par libelléLigne.

Après l'appel de cette commande, lorsqu'un utilisateur sélectionne la ligne de menu en mode Développement ou Application, la méthode méthode est appelée. Typiquement, cette méthode affiche une boîte de dialogue qui fournit des informations sur les versions de votre base.

Cette commande est utilisable avec 4D (tous modes), 4D Desktop et 4D Server. Son exécution sur le poste serveur provoque la création d'un nouveau process.

Exemples

(1) L'exemple suivant remplace la commande de menu **A propos** par la commande de menu **A propos du programmateur**. La méthode A PROPOS affiche une fenêtre d'A propos personnalisée :

```
APPELER SUR A PROPOS("A propos du programmateur";"A PROPOS")
```

(2) L'exemple suivant réinitialise la commande de menu d'A propos de 4D :

```
APPELER SUR A PROPOS("A propos de 4D®";"")
```

BEEP

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande BEEP provoque l'émission d'un bip sonore. Votre PC ou votre Macintosh peut émettre un autre son qu'un bip en fonction du son sélectionné dans le tableau de bord de contrôle du son.

ATTENTION : N'appelez pas la commande BEEP à partir d'un process de connexion Web car le bip sonore se produira sur le poste serveur Web 4D et non sur le poste du navigateurWeb.

Exemple

Dans l'exemple suivant, un bip est émis et une alerte affichée lorsqu'aucun enregistrement n'est trouvé par une recherche :

```
CHERCHER([Clients];[Clients]Nom=$vsNomAChercher)
Si (Enregistrements trouves([Clients])=0)
    BEEP
    ALERTE("Il n'y a aucun client de ce nom.")
Fin de si
```

Référence

JOUER SON.

CACHER BARRE DE MENUS

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande CACHER BARRE DE MENUS rend invisible la barre de menus.

Si la barre de menus était déjà cachée, la commande est sans effet.

Exemple

La méthode suivante passe un enregistrement en plein écran (sous Mac OS) jusqu'à ce que l'utilisateur clique sur le bouton de la souris :

```
CACHER BARRE OUTILS
CACHER BARRE DE MENUS
Creer fenetre(-1;-1;1+Largeur ecran;1+Hauteur ecran;Autre boîte de dialogue modale)
FORMULAIRE ENTREE([Tableaux];"Plein écran 800")
AFFICHER ENREGISTREMENT([Tableaux])
Repeter
   POSITION SOURIS($vlX;$vlY;$vlBouton)
Jusque($vlBouton#0)
FERMER FENETRE
AFFICHER BARRE DE MENUS
AFFICHER BARRE OUTILS
```

Note : Sous Windows, la taille de la fenêtre sera limitée par celle de la fenêtre de l'application.

Référence

AFFICHER BARRE DE MENUS, AFFICHER BARRE OUTILS, CACHER BARRE OUTILS.

CACHER BARRE OUTILS

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande CACHER BARRE OUTILS rend invisible la barre d'outils. Si la barre d'outils était déjà cachée, CACHER BARRE OUTILS ne fait rien.

Référence

AFFICHER BARRE DE MENUS, AFFICHER BARRE OUTILS, CACHER BARRE DE MENUS.

Note de compatibilité : Cette commande a été conservée pour des raisons de compatibilité uniquement car elle s'appuie sur des mécanismes désormais obsolètes (ressources 'CURS' de Mac OS). Il est déconseillé de l'utiliser dans de nouveaux développements.

CHANGER POINTEUR SOURIS {(curseur)}

Paramètre	Type	Description
curseur	Numérique →	Numéro de ressource Mac OS de curseur

Description

La commande CHANGER POINTEUR SOURIS remplace le pointeur (graphique) de la souris par celui qui est stocké dans la ressource type Mac OS 'CURS' dont vous avez passé le numéro d'ID dans le paramètre curseur.

Si vous ne passez pas ce paramètre, le pointeur de la souris (re)devient la flèche standard.

Voici à titre d'exemple les numéros des curseurs basiques pouvant être passés dans le paramètre curseur :

1 

2 

3 

4 

DEFILER LIGNES ({*; }objet{; position{; *}})

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une table ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Table ou variable (si * est omis)
position	Entier long	→	Numéro de ligne à afficher
*	*	→	Afficher la ligne en première position après défilement

Description

La commande DEFILER LIGNES permet de faire défiler les lignes d'un sous-formulaire, d'un formulaire liste affiché via la commande MODIFIER SELECTION ou VISUALISER SELECTION, d'une liste hiérarchique ou encore d'un objet de type List box de manière à afficher le premier enregistrement/la première ligne sélectionné(e) ou un enregistrement/une ligne spécifique.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre objet est le nom d'un objet de type sous-formulaire, liste hiérarchique ou List box (dans ce cas, passez une chaîne dans objet). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une table (table du formulaire liste ou du sous-formulaire) ou une variable (RefList de liste hiérarchique ou list box).

Si vous ne passez pas le paramètre position, la commande provoque le défilement vertical des lignes de la liste de manière à ce que la première ligne sélectionnée (surlignée) dans la liste soit visible. Si aucune ligne n'est sélectionnée, la commande ne fait rien. Si au moins une ligne sélectionnée est déjà visible, la commande ne fait rien.
Le paramètre position permet de spécifier le numéro de la ligne à afficher. Si vous passez ce paramètre, la commande provoque le défilement vertical des lignes de la liste de manière à ce que la ligne désignée soit visible (qu'elle soit surlignée ou non). Si la ligne est déjà visible, la commande ne fait rien. Pour les formulaires liste et les sous-formulaires, ce numéro correspond au numéro d'un enregistrement parmi la sélection courante, c'est-à-dire sa position. Dans le cas des listes hiérarchiques, la commande tient compte de l'état déployé/contracté des éléments. Pour les list box, ce numéro correspond au numéro de la ligne parmi toutes les lignes de l'objet (y compris les lignes éventuellement cachées). Si le numéro passé dans position correspond à une ligne masquée dans la list box, la commande affiche la première ligne visible suivante.

Si vous passez le second paramètre optionnel *, la ligne rendue visible par la commande (si la liste a effectivement défilé) sera placée en première position de la liste.

Note : La commande MARQUER ENREGISTREMENTS comporte un paramètre * facultatif permettant de déléguer la gestion du défilement dans les formulaires à la commande DEFILER LIGNES.

Référence

MARQUER ENREGISTREMENTS, SELECTIONNER LIGNE LISTBOX.

Note de compatibilité
Cette commande est conservée pour des raisons de compatibilité uniquement. Dans les bases de données créées à partir de la version 2004 de 4D, l'interface de plate-forme est gérée automatiquement par le programme et cette commande est ignorée. Elle peut toujours être utilisée dans les bases de données converties, toutefois il est conseillé d'activer le nouveau mode de gestion via les Préférences de l'application (option "Système").

FIXER INTERFACE (interface)

Paramètre	Type		Description
interface	Numérique	→	Nouvelle interface de plate-forme : -1 Plate-forme automatique 0 Mac OS 7 1 Windows 3.11, NT 3.51 2 Windows 9x 3 Mac OS 9 4 Thème Mac

Description

La commande FIXER INTERFACE définit l'interface de plate-forme utilisée pour afficher les formulaires.

Vous passez dans le paramètre interface une des constantes prédéfinies suivantes :

Constante	Type	Valeur
Plate forme automatique	Entier long	-1
Mac OS 7	Entier long	0
Windows 3.11, NT 3.51	Entier long	1
Windows 9x	Entier long	2
Mac OS 9	Entier long	3
Thème Mac	Entier long	4

Note : La constante Thème Mac permet d'utiliser l'interface définie dans le Gestionnaire d'apparence. Ce gestionnaire n'existe que sous Mac OS. Lorsqu'une base définie en Thème Mac est affichée sous Windows, l'interface "Windows 9x" est appliquée.

Si la valeur que vous passez ne modifie pas l'interface de plate-forme courante, la commande ne fait rien.

Note : L'interface de plate-forme peut également être modifiée dans la boîte de dialogue des Préférences.

Exemple

Avec 4D Server, vous voulez que les postes 4D Client Macintosh et Windows utilisent simultanément des interfaces de plate-formes différentes. Pour cela, vous pouvez appeler la commande FIXER INTERFACE dans la Méthode base Sur ouverture :

```
    ` Partons de l'hypothèse que les préférences des utilisateurs sont stockées dans une
    ` table [Préférences]
    ` Cherchons l'enregistrement correspondant à l'utilisateur courant
CHERCHER([Préférences];[Préférences]Utilisateur=Utilisateur courant)
Si (Enregistrements trouves([Préférences])=0)
        ` S'il n'est pas trouvé, utilisons les préférences par défaut
    CHERCHER([Préférences];[Préférences]Utilisateur="Par défaut")
Fin de si
    ` Fixons l'interface de plate-forme en fonction des préférences de l'utilisateur
FIXER INTERFACE ([Préférences]Interface plate-forme)
```

Référence

Lire interface.

FIXER TITRES CHAMPS (table; titresChamps; numChamps{; *})

Paramètre	Type		Description
table	Table	→	Table dont vous voulez redéfinir les titres des champs
titresChamps	Tableau Alpha	→	Nouveaux titres des champs
numChamps	Tableau Num	→	Numéros des champs
*		→	Utiliser les noms personnalisés dans l'éditeur de formules

Description

FIXER TITRES CHAMPS vous permet de masquer, renommer et réordonner les champs d'une table de votre base lorsqu'ils apparaissent dans les éditeurs standard de 4D, tel que l'éditeur de recherches, en mode Application (plus précisément, lorsque les éditeurs sont appelés via des commandes du langage de 4D).
Cette commande vous permet également de modifier les libellés des champs apparaissant dans vos formulaires, si vous avez utilisé des libellés dynamiques. Pour plus d'informations sur l'insertion de libellés de tables et de champs dynamiques dans les formulaires, reportez-vous au manuel *Mode Développement*.

Les tableaux titresChamps et numChamps doivent être synchronisés. Dans le tableau titresChamps, vous passez les noms des champs tels que vous voulez qu'ils apparaissent. Les champs s'afficheront dans l'ordre défini par ces tableaux. Dans chaque élément du tableau numChamps, vous passez le numéro de la table qui correspond au nom, nouveau ou ancien, du champ, et ce dans le même numéro d'élément que dans le tableau titresChamps.
Si vous voulez masquer un champ, il suffit de ne pas le passer dans les tableaux. Vous avez, par exemple, une table comprenant les champs F, G et H, créés dans cet ordre. Vous voulez que ces champs apparaissent comme M, N et O. De plus, vous ne voulez pas faire apparaître le champ N. Enfin, vous voulez que les tables soient dans l'ordre O et M. Pour cela, vous passez dans le paramètre titresChamps un tableau contenant deux éléments, O et M, et vous passez dans le paramètre numChamps un tableau contenant deux éléments, 3 et 1.

Le paramètre facultatif * vous permet d'indiquer si les noms personnalisés définis à l'aide de cette commande peuvent être ou non utilisés dans les formules de 4D.
• Par défaut, lorsque ce paramètre est omis, les formules exécutées dans 4D ne peuvent pas utiliser ces noms personnalisés ; il est nécessaire d'utiliser les noms de champs véritables.
• Si le paramètre * est passé, les noms définis par cette commande peuvent être utilisés dans les formules exécutées par 4D. **Attention dans ce cas**, les noms personnalisés ne doivent pas contenir de caractères "interdits" par l'interpréteur du langage de 4D, tels que -?*! (pour plus d'informations, reportez-vous à la section "Nommer les objets du langage 4D").

Note : Au niveau de l'éditeur de formules, l'exécution de cette commande sans le paramètre * ne modifie pas les éventuels paramétrages effectués précédemment avec le paramètre * . Autrement dit, l'éditeur de formules affiche toujours le nom personnalisé défini via le dernier appel de la commande avec le paramètre *.

FIXER TITRES CHAMPS ne modifie pas la structure de votre base. Elle n'affecte que l'affichage ultérieur des éditeurs standard de 4D et des formulaires comportant des libellés dynamiques lorsqu'ils sont appelés via une commande de langage (la structure réelle de la base est affichée lorsque l'éditeur ou le formulaire est appelé depuis une commande de menu en mode Développement). L'aire de validité de la commande FIXER TITRES CHAMPS est la session. L'avantage, en Client/Serveur, est que plusieurs 4D distants peuvent “voir” simultanément votre structure d'une manière différente. Vous pouvez appeler FIXER TITRES CHAMPS autant de fois que vous voulez.

La commande FIXER TITRES CHAMPS est utile dans les cas suivants :
• Traduction dynamique de votre base.
• Affichage des champs dans l'ordre et avec les noms que vous voulez, indépendamment de leurs définitions.
• Affichage des champs suivant l'identité ou les privilèges d'un utilisateur.

ATTENTION :
• FIXER TITRES CHAMPS n'annule pas l'effet de la propriété Invisible d'un champ. Si vous avez défini un champ en tant qu'invisible au niveau de la structure, il n'apparaîtra pas en mode Application même s'il est spécifié dans FIXER TITRES CHAMPS.
• Les plug-ins accèdent toujours à la structure "virtuelle" telle que définie par cette commande.

Exemple

Reportez-vous à l'exemple de la commande FIXER TITRES TABLES.

Référence

FIXER TITRES TABLES, Lire numero dernier champ, LIRE TITRES CHAMPS, Nom du champ.

FIXER TITRES TABLES (titresTables; numTables{; *})

Paramètre	Type		Description
titresTables	Tableau Alpha	→	Noms des tables tels qu'ils doivent apparaître
numTables	Tableau Num	→	Numéros des tables
*		→	Utiliser les noms personnalisés dans l'éditeur de formules

Description

FIXER TITRES TABLES vous permet de masquer, renommer et réordonner les tables de votre base lorsqu'elles apparaissent dans les éditeurs standard de 4D, comme l'éditeur de recherches, en mode Application (plus précisément, lorsque les éditeurs sont appelés via des commandes du langage de 4D).
Cette commande vous permet également de modifier les libellés des tables apparaissant dans vos formulaires, si vous avez utilisé des libellés dynamiques. Pour plus d'informations sur l'insertion de libellés de tables et de champs dynamiques dans les formulaires, reportez-vous au manuel *Mode Développement*.

Les tableaux titresTables et numTables doivent être synchronisés. Dans le tableau titresTables, vous passez les noms des tables tels que vous voulez qu'ils apparaissent. Les tables s'afficheront dans l'ordre défini par ces tableaux. Dans chaque élément du tableau numTables, passez le numéro de la table qui correspond au nom, nouveau ou ancien, de la table, et ce dans le même numéro d'élément que dans le tableau titresTables.
Si vous voulez masquer une table, ne la mettez pas dans les tableaux. Vous avez, par exemple, une base comprenant les tables A, B et C, créées dans cet ordre. Vous voulez que ces tables apparaissent sous les noms X, Y et Z. De plus, vous ne voulez pas faire apparaître la table B. Enfin, vous voulez que les tables soient dans l'ordre Z et X. Pour cela, vous passez dans le paramètre titresTables un tableau à deux éléments, Z et X, et vous passez dans le paramètre numTables un tableau à deux éléments, 3 et 1.

Le paramètre facultatif * vous permet d'indiquer si les noms personnalisés définis à l'aide de cette commande peuvent être ou non utilisés dans les formules de 4D.
• Par défaut, lorsque ce paramètre est omis, les formules exécutées dans 4D ne peuvent pas utiliser ces noms personnalisés ; il est nécessaire d'utiliser les noms de tables véritables.

• Si le paramètre * est passé, les noms définis par cette commande peuvent être utilisés dans les formules exécutées par 4D. **Attention dans ce cas**, les noms personnalisés ne doivent pas contenir de caractères “interdits” par l’interpréteur du langage de 4D, tels que -?*! (pour plus d'informations, reportez-vous à la section “Nommer les objets du langage 4D”).

Note : Au niveau de l'éditeur de formules, l'exécution de cette commande sans le paramètre * ne modifie pas les éventuels paramétrages effectués précédemment avec le paramètre * . Autrement dit, l'éditeur de formules affiche toujours le nom personnalisé défini via le dernier appel de la commande avec le paramètre *.

FIXER TITRES TABLES ne modifie pas la structure de votre base. Cette commande n'affecte que l'utilisation ultérieure des éditeurs standard de 4D et des formulaires comportant des libellés dynamiques lorsqu'ils sont appelés via une commande de langage (la structure réelle de la base est affichée lorsque l'éditeur ou le formulaire est appelé depuis une commande de menu en mode Développement). L'aire de validité de la commande FIXER TITRES TABLES est la session. L'avantage, en Client/Serveur, est que plusieurs postes 4D Client peuvent “voir” simultanément votre structure d'une manière différente. Vous pouvez appeler FIXER TITRES TABLES autant de fois que vous voulez.

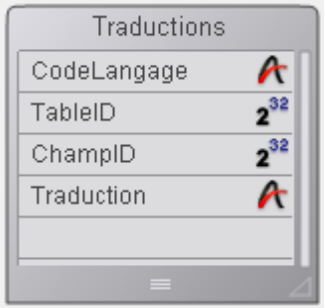
La commande FIXER TITRES TABLES est utile dans les cas suivants :
• Traduction dynamique de votre base.
• Affichage des tables dans l'ordre et avec les noms que vous voulez, indépendamment de leur définition.
• Affichage des tables d'une façon qui dépend de l'identité ou des privilèges d'un utilisateur.

Notes :
• FIXER TITRES TABLES n'annule pas l'effet de la propriété Invisible d'une table. Si vous avez défini une table en tant qu'invisible au niveau de la structure, elle n'apparaîtra pas en mode Application, même si elle est spécifiée dans FIXER TITRES TABLES.
• Les plug-ins accèdent toujours à la structure "virtuelle" telle que définie par cette commande.

Exemple

• Vous développez une application 4D destinée au marché international. Vous avez donc besoin de prendre en compte les nécessités de traduction et de localisation. Pour les éditeurs standard de 4D qui apparaissent en mode Application et vos formulaires utilisant des libellés dynamiques, vous pouvez traiter cette question en utilisant une table [Traductions] et quelques méthodes pour créer et utiliser les traductions pour chaque langue que vous voulez.

• Dans votre base, vous créez la table suivante :



• Ensuite, créez la méthode projet traduire_TABLES_ET_CHAMPS ci-dessous. Cette méthode analyse la structure de votre base dans la table [Traductions] et crée les enregistrements correspondant à la langue passée comme paramètre.

```
` méthode projet traduire_TABLES_ET_CHAMPS
` traduire_TABLES_ET_CHAMPS ( Texte)
` traduire_TABLES_ET_CHAMPS ( CodeLangue )

C_TEXTE($1) `code de la langue
C_ENTIER LONG($vlTable;$vlChamp)
C_TEXTE($Langue)
$Langue:=$1

Boucle ($vlTable;1;Lire numero derniere table)  ` Passer sur chaque table
   Si ($vlTable#(Table(->[Traductions])))  `Ne pas traduire la table des traductions
         ` Vérifier s'il existe une traduction du nom de la table pour la langue spécifiée
      CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)
         `langue souhaitée
      CHERCHER([Traductions]; & ;[Traductions]TableID=$vlTable;*)  `numero de table
      CHERCHER([Traductions]; & ;[Traductions]ChampID=0)
            `numero de champ = 0 signifie que c'est un nom de table
      Si (Est un numero de table valide($vlTable))  `vérifier que la table existe encore
         Si (Enregistrements trouves([Traductions])=0)
               ` Sinon, créer l'enregistrement
            CREER ENREGISTREMENT([Traductions])
            [Traductions]CodeLangage:=$Langue
            [Traductions]TableID:=$vlTable
            [Traductions]ChampID:=0
```

```
                ` Le nom de la table traduit aura besoin d'être saisi
            [Traductions]Traduction:=Nom de la table($vlTable)+" en "+$Langue
            STOCKER ENREGISTREMENT([Traductions])
        Fin de si

        Boucle ($vlChamp;1;Lire numero dernier champ($vlTable))
            ` Vérifier s'il existe une traduction pour le nom du champ dans la
            ` langue spécifiée
            CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)
            `langue souhaitée
            CHERCHER([Traductions]; & ;[Traductions]TableID=$vlTable;*)
            `numéro de table
            CHERCHER([Traductions]; & ;[Traductions]ChampID=$vlChamp)
            `numéro de champ
            Si (Est un numero de champ valide($vlTable;$vlChamp))
                Si (Enregistrements trouves([Traductions])=0)
                    ` Sinon, créer l'enregistrement
                    CREER ENREGISTREMENT([Traductions])
                    [Traductions]CodeLangage:=$Langue
                    [Traductions]TableID:=$vlTable
                    [Traductions]ChampID:=$vlChamp
                    ` Le nom du champ traduit aura besoin d'être saisi
                    [Traductions]Traduction:=Nom du champ($vlTable;$vlChamp)+
                                                                " en "+$Langue
                    STOCKER ENREGISTREMENT([Traductions])
                Fin de si
            Sinon
                Si (Enregistrements trouves([Traductions])#0)
                    ` si le champ n'existe plus, on supprime la traduction
                    SUPPRIMER ENREGISTREMENT([Traductions])
                Fin de si
            Fin de si
        Fin de boucle
    Sinon
        Si (Enregistrements trouves([Traductions])#0)
            ` si la table n'existe plus, on supprime la traduction
            SUPPRIMER ENREGISTREMENT([Traductions])
        Fin de si
    Fin de si
Fin de si
Fin de boucle
```

• Si maintenant vous exécutez la ligne suivante, vous pouvez créer autant d'enregistrements qu'il vous faut pour la traduction espagnole de vos tables et champs :

traduire_TABLES_ET_CHAMPS ("Espagnol")

• Une fois que cette ligne de code est appelée, vous pouvez saisir une traduction dans le champ [Traductions]NomTraduit pour chacun des nouveaux enregistrements.

• Enfin, chaque fois que vous voulez afficher en espagnol les éditeurs standard de 4D ou les formulaires avec libellés dynamiques, vous exécutez la ligne suivante :

TABLES_ET_CHAMPS_LOCALISES ("Espagnol")

La méthode projet TABLES_ET_CHAMPS_LOCALISES est la suivante :

```
      ` Méthode objet TABLES_ET_CHAMPS_LOCALISES
      ` TABLES_ET_CHAMPS_LOCALISES ( Texte)
      ` TABLES_ET_CHAMPS_LOCALISES ( CodeLangue )

C_TEXTE($1)  `Code de la langue
C_ENTIER LONG($vlTable;$vlChamp)
C_TEXTE($Langue)
C_ENTIER LONG($vlNumTable;$vlNumChamp)
$Langue:=$1

   `Mise à jour des noms de table
TABLEAU TEXTE($asNoms;0)
   ` Initialiser les tableaux pour FIXER TITRES TABLES et FIXER TITRES CHAMPS
TABLEAU ENTIER($aiNuméros;0)
CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)
CHERCHER([Traductions]; & ;[Traductions]ChampID=0)  `noms de table donc
SELECTION VERS TABLEAU([Traductions]Traduction;$asNoms;[Traductions]TableID;
                                                                $aiNuméros)
FIXER TITRES TABLES($asNoms;$aiNuméros)

   `Mise à jour des noms de champs
$vlNumTable:=Lire numero derniere table  ` Obtenir le nombre de tables dans la base
Boucle ($vlTable;1;$vlNumTable)  ` Passer sur chaque table
   Si (Est un numero de table valide($vlTable))
      CHERCHER([Traductions];[Traductions]CodeLangage=$Langue;*)
            CHERCHER([Traductions]; & ;[Traductions]TableID=$vlTable;*)
```

```
            CHERCHER([Traductions]; & ;[Traductions]ChampID#0)
                `évite le zero qui sert au nom de la table
            SELECTION VERS TABLEAU([Traductions]Traduction;$asNoms;
                                                    [Traductions]ChampID;$aiNumeros)
            FIXER TITRES CHAMPS(Table($vlTable)->;$asNoms;$aiNumeros)
        Fin de si
    Fin de boucle
```

• Notez que de nouvelles traductions peuvent être effectuées dans la base sans modification de code ni recompilation.

Référence

FIXER TITRES CHAMPS, Lire numero derniere table, Nom de la table.

GENERER CLIC (sourisX; sourisY{; process}{; *})

Paramètre	Type		Description
sourisX	Numérique	→	Coordonnée horizontale
sourisY	Numérique	→	Coordonnée verticale
process	Numérique	→	Numéro de référence du process de destination ou File d'attente des événements de l'application si ce paramètre est omis ou si vous passez 0
*		→	Si spécifié, utiliser le système de coordonnées globales Si omis, utiliser le système de coordonnées locales

Description

La commande GENERER CLIC simule un clic souris. Elle produit les mêmes effets que lorsque l'utilisateur clique réellement avec le bouton de la souris.

Vous passez les coordonnées horizontale et verticale du clic dans sourisX et sourisY. Si vous passez le paramètre *, vous exprimez ces coordonnées par rapport à l'écran. Si vous omettez le paramètre *, vous exprimez ces coordonnées par rapport à la fenêtre de premier plan du process dont le numéro est passé dans process.

Si vous passez le paramètre process, le clic est envoyé au process dont vous avez passé le numéro. Si vous passez 0 (zéro) ou si vous omettez ce paramètre, le clic est envoyé au niveau de l'application et le gestionnaire de 4D l'affectera au process approprié.

Référence

GENERER EVENEMENT, GENERER FRAPPE CLAVIER.

GENERER EVENEMENT (quoi; message; quand; sourisX; sourisY; modifiers{; process})

Paramètre	Type		Description
quoi	Numérique	→	Type d'événement
message	Numérique	→	Message de l'événement
quand	Numérique	→	Moment de l'événement exprimé en ticks
sourisX	Numérique	→	Coordonnée horizontale de la souris
sourisY	Numérique	→	Coordonnée verticale de la souris
modifiers	Numérique	→	Etat des touches Modifier
process	Numérique	→	Numéro de référence du process de destination ou File d'attente des événements de l'application si ce paramètre est omis ou si vous passez 0

Description

La commande GENERER EVENEMENT simule un événement (clavier ou souris). Elle produit les mêmes effets que lorsque l'utilisateur agit réellement par l'intermédiaire du clavier ou de la souris.

Vous devez passer une des constantes prédéfinies suivantes dans le paramètre quoi :

Constante	Type	Valeur
Bouton souris enfoncé	Entier long	1
Bouton souris relâché	Entier long	2
Touche enfoncée	Entier long	3
Touche relâchée	Entier long	4
Répétition touche	Entier long	5

Si l'événement est lié à la souris, passez 0 (zéro) dans le paramètre message. Si l'événement est lié au clavier, passez dans message le code du caractère simulé.

Généralement, vous passez la valeur retournée par la fonction Nombre de ticks dans quand.

Si l'événement est lié à la souris, passez les coordonnées horizontale et verticale du clic dans sourisX et sourisY.

Dans le paramètre modifiers, vous devez passer une constante ou une combinaison de constantes du thème **Evénements (Modifiers)**. Par exemple, pour simuler la touche Majuscule, passez la valeur Bit touche majuscule.

Si vous passez le paramètre process, l'événement est envoyé au process dont vous avez passé le numéro. Si vous passez 0 (zéro) ou si vous omettez ce paramètre, l'événement est envoyé au niveau de l'application et le gestionnaire de 4D l'affectera au process approprié.

Référence

GENERER CLIC, GENERER FRAPPE CLAVIER.

GENERER FRAPPE CLAVIER (code{; modifiers{; process}})

Paramètre	Type		Description
code	Numérique	→	Code d'un caractère ou code de touche de fonction
modifiers	Numérique	→	Etat des touches Modifier
process	Numérique	→	Numéro de référence du process de destination ou File d'attente des événements de l'application si paramètre omis ou égal à 0

Description

La commande GENERER FRAPPE CLAVIER simule la frappe d'une touche sur le clavier. Elle produit les mêmes effets que lorsque l'utilisateur tape réellement un caractère au clavier.

Vous passez le code du caractère dans le paramètre code.

Si vous n'utilisez pas le paramètre modifiers, aucun "modifier" (Majuscule, Option, etc...) n'est simulé. Si vous utilisez le paramètre modifiers, vous devez passer une constante ou une combinaison de constantes du thème **Evénements (Modifiers)**. Par exemple, pour simuler la touche Majuscule, passez la valeur Masque touche majuscule.

Si vous passez le paramètre process, la frappe clavier est envoyée au process dont le numéro de référence est spécifié. Si vous passez 0 (zéro) dans ce paramètre ou si vous l'omettez, la frappe clavier est envoyée au niveau de l'application et le gestionnaire de 4D l'affectera au process approprié.

Exemple

Reportez-vous à l'exemple de la fonction Chercher process.

Référence

GENERER CLIC, GENERER EVENEMENT.

Hauteur barre outils → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Hauteur (exprimée en pixels) de la barre d'outils ou 0 si la barre d'outils n'est pas affichée

Description

La commande Hauteur barre outils retourne la hauteur de la barre d'outils de 4D, exprimée en pixels. Si la barre d'outils n'est pas affichée, la commande retourne 0.

Référence

AFFICHER BARRE OUTILS, CACHER BARRE OUTILS, Hauteur barre de menus.

INVERSER FOND ({*; }textVar | textChp)

Paramètre	Type		Description
*	*	→	Si spécifié, on passe le nom de l'objet (chaîne) Si omis, on passe un champ ou une variable
textVar | textChp	Variable | Champ	→	Variable ou champ de type texte dont le fond doit être inversé

Description

INVERSER FOND permet d'inverser textVar ou textChp dans un formulaire.

La portée de cette commande est le formulaire en cours d'utilisation.

Si vous passez le paramètre optionnel *, vous indiquez que le second paramètre désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables texte uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Vous pouvez utiliser INVERSER FOND lors d'un affichage à l'écran ou lors d'une impression sur une imprimante matricielle. Une imprimante postscript ne permet pas d'inverser le fond.

Vous ne pouvez pas non plus inverser de variable dans un formulaire sortie. Evitez d'utiliser INVERSER FOND avec une variable saisissable : la saisie de caractères effacera partiellement seulement l'inversion.

Exemple

L'exemple suivant est la méthode objet d'une variable dans un formulaire entrée. La méthode teste la valeur d'un champ. Si elle est positive, la méthode ne fait rien. Si elle est négative, la variable est inversée dans le formulaire :

```
vMontant := [Comptes]Montant ` Assignation de la valeur du champ à la variable
Si (vMontant < 0) ` Si le montant est négatif…
    INVERSER FOND (vMontant) ` Inverser le fond de la variable
Fin de si
```

Note : Cette commande, créée à l'origine pour les interfaces en noir et blanc, est désormais rarement utilisée. Pour signaler ou mettre en avant un champ ou une variable dont la valeur est par exemple incorrecte, un système de couleurs est généralement utilisé.

Référence

CHOIX COULEUR, FIXER COULEURS RVB.

JOUER SON (nomObjet{; canal})

Paramètre	Type		Description
nomObjet	Alpha	→	Nom de son Windows : extension de fichier .WAV, .MID ou .AVI Toute plate-forme : ressource Mac OS 'snd' ou chaîne vide pour stopper un son asynchrone
canal	Numérique	→	Si passé : canal de sortie et exécution asynchrone Si omis : exécution synchrone

Description

Sous Windows, la commande JOUER SON permet de jouer des fichiers Windows de sons (fichiers .WAV), MIDI (fichiers .MID) ou vidéo (fichiers .AVI). Vous passez le chemin d'accès complet du fichier que vous voulez jouer dans nomObjet.

NOTE : Vous ne pouvez pas jouer des fichiers ou des objets multimédia en mode asynchrone. Pour cela, utilisez les services OLE.

Sous Mac OS (ou sous Windows dans certaines conditions, cf. paragraphe ci-dessous), JOUER SON joue la ressource son nomObjet.

Le paramètre canal spécifie le canal de sortie de synthétiseur Macintosh. Si le canal n'est pas spécifié, le canal est utilisé pour des sons digitalisés simples et est synchrone. Synchrone signifie que tous les traitements s'arrêtent jusqu'à ce que le son soit entièrement joué. Si canal est égal à 0, le canal est utilisé pour des sons digitalisés simples et est asynchrone. Asynchrone signifie que le traitement ne s'arrête pas et que le son est joué en tâche de fond.

Pour stopper un son synchrone, il faut exécuter l'instruction suivante :

```
JOUER SON ("";0)
```

Si vous avez une base qui fonctionne à la fois sur Macintosh et sur PC, vous pouvez jouer des sons Macintosh sous Windows. Pour cela :
• Sous Mac OS, à l'aide d'un éditeur de ressources tel que ResEdit™ ou Resorcerer™, copiez les ressources 'snd ' nécessaires dans la "resource fork" du fichier de structure de la base.
• Transportez la base de Macintosh à Windows à l'aide de 4D Transporter.

Note importante : La version Windows de 4D ne joue pas les sons Macintosh compressés à l'aide de MACE. Si votre ressource 'snd' Macintosh ne se joue pas sous Windows, déterminez si le son est conforme aux conditions suivantes :

champ de ressource snd	Valeur (en hexadécimal)
Version	0x0001
NbSynth	0x0001
SynthResID	0x0005
SynthInitOptions	0x000000A0
NbSoundCommand	0x0001
FirstCommand	0x8051

Vous pouvez vérifier les valeurs internes d'une ressource 'snd ' à l'aide de Resorcerer™.

Exemples

(1) L'exemple suivant montre comment jouer un fichier vidéo de votre choix sous Windows :

```
$DocRéf := Ouvrir document ( ""; "AVI")
Si (OK=1)
    FERMER DOCUMENT($DocRéf)
    JOUER SON (Document)
Fin de si
```

(2) L'exemple suivant se trouve dans une méthode de démarrage. Ce son est joué lors de l'ouverture de la base sous Mac OS :

```
JOUER SON ("Bienvenue")  ` Jouer le son de bienvenue
```

Référence

BEEP.

Lire interface

version 2004 (Modifiée)

Note de compatibilité
Cette commande est conservée pour des raisons de compatibilité uniquement. Dans les bases de données créées à partir de la version 2004 de 4D, l'interface de plate-forme est gérée automatiquement par le programme et cette commande est ignorée. Elle peut toujours être utilisée dans les bases de données converties, toutefois il est conseillé d'activer le nouveau mode de gestion de l'interface via les Préférences de l'application (option "Système").

Lire interface → Numérique

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Numérique	←	Interface de plate-forme en cours d'utilisation

Description

La fonction Lire interface retourne une valeur numérique indiquant l'interface de plate-forme utilisée pour afficher les formulaires.

Cette fonction peut retourner une des constantes prédéfinies suivantes, placées dans le thème “Interfaces de plate-forme” :

Constante	Type	Valeur
Plate forme automatique	Entier long	-1
Mac OS 7	Entier long	0
Windows 3.11, NT 3.51	Entier long	1
Windows 9x	Entier long	2
Mac OS 9	Entier long	3
Thème Mac	Entier long	4

Vous pouvez modifier l'interface de plate-forme à l'aide de la commande FIXER INTERFACE ou en utilisant la boîte de dialogue des Préférences.

Référence

FIXER INTERFACE.

LIRE TITRES CHAMPS (laTable; titresChamps; numChamps)

Paramètre	Type		Description
laTable	Table	→	Table dont vous souhaitez connaître les noms des champs
titresChamps	Tableau Texte	←	Noms courants des champs
numChamps	Tableau Entier long	←	Numéros des champs

Description

La commande LIRE TITRES CHAMPS remplit les tableaux titresChamps et numChamps avec les noms et les numéros des champs de laTable désignée. Le contenu des deux tableaux est synchronisé.

Si la commande FIXER TITRES CHAMPS a été appelée au cours de la session, LIRE TITRES CHAMPS retourne uniquement les noms “modifiés” et les numéros des champs ayant été définis via cette commande.
Sinon, LIRE TITRES CHAMPS retourne le nom défini dans la fenêtre de Structure de tous les champs de la base.
Dans les deux cas, la commande ne retourne pas les champs déclarés invisibles.

Référence

FIXER TITRES CHAMPS, LIRE TITRES TABLES.

LIRE TITRES TABLES (titresTables; numTables)

Paramètre	Type		Description
titresTables	Tableau Texte	←	Noms courants des tables
numTables	Tableau Entier long	←	Numéros des tables

Description

La commande LIRE TITRES TABLES remplit les tableaux titresTables et numTables avec les noms et les numéros des tables de la base définis dans la fenêtre de Structure ou via la commande FIXER TITRES TABLES. Le contenu des deux tableaux est synchronisé.

Si la commande FIXER TITRES TABLES a été appelée lors de la session, LIRE TITRES TABLES retourne uniquement les noms “modifiés” et les numéros des tables ayant été définies via cette commande.
Sinon, LIRE TITRES TABLES retourne le nom défini dans la fenêtre de Structure de toutes les tables de la base.
Dans les deux cas, la commande ne retourne pas les tables déclarées invisibles.

Référence

FIXER TITRES TABLES, LIRE TITRES CHAMPS.

Macintosh commande enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Commande Macintosh ou Etat de la touche Ctrl Windows

Description

Macintosh commande enfoncee retourne Vrai si la touche **Commande** Macintosh est enfoncée.

Note : Lorsqu'elle est appelée sous Windows, la fonction Macintosh commande enfoncee retourne Vrai si la touche **Ctrl** Windows est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande Majuscule enfoncee.

Référence

Macintosh control enfoncee, Macintosh option enfoncee, Majuscule enfoncee, Verrouillage majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

Macintosh control enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Control du Macintosh

Description

Macintosh control enfoncee retourne Vrai si la touche **Control** du Macintosh est enfoncée.

Note : Lorsqu'elle est appelée sous Windows, la fonction Macintosh control enfoncee retourne toujours Faux. Cette touche Macintosh n'a pas d'équivalent sous Windows.

Exemple

Reportez-vous à l'exemple de la commande Majuscule enfoncee.

Référence

Macintosh commande enfoncee, Macintosh option enfoncee, Majuscule enfoncee, Verrouillage majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

Macintosh option enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Option Macintosh ou Etat de la touche Alt Windows

Description

Macintosh option enfoncee retourne Vrai si la touche **Option** Macintosh est enfoncée.

Note : Lorsqu'elle est appelée sous Windows, la fonction Macintosh option enfoncee retourne Vrai si la touche **Alt** Windows est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande Majuscule enfoncee.

Référence

Macintosh commande enfoncee, Macintosh control enfoncee, Majuscule enfoncee, Verrouillage majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

Majuscule enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Majuscule

Description

Majuscule enfoncee retourne Vrai si la touche **Majuscule** est enfoncée.

Exemple

La méthode objet du bouton bUnBouton effectue des actions différentes en fonction de la ou des touche(s) de modification enfoncée(s) au moment du clic :

```
` Méthode objet bUnBouton
Au cas ou
        ` Diverses autres combinaisons de touches peuvent être testées ici
        ` ...
   : (Majuscule enfoncee & Windows Ctrl enfoncee)
        ` Les touches Majuscule et Ctrl Windows (ou Commande Mac OS)
        ` sont enfoncées
      FAIRE ACTION1
        ` ...
   : (Majuscule enfoncee)
        ` Seule Majuscule est enfoncée
      FAIRE ACTION2
        ` ...
```

```
: (Windows Ctrl enfoncee)
      ` Seule Ctrl Windows (ou Commande Mac OS) est enfoncée
   FAIRE ACTION3
      ` ...
      ` D'autres touches peuvent être testées individuellement ici
      ` ...
Fin de cas
```

Référence

Macintosh commande enfoncee, Macintosh control enfoncee, Macintosh option enfoncee, Verrouillage majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

Objet focus → Pointeur

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Pointeur	←	Pointeur vers l'objet ayant le focus

Description

Objet focus retourne un pointeur vers l'objet ayant le focus dans le formulaire courant. Si aucun objet n'a le focus, la commande retourne Nil. Vous pouvez utiliser Objet focus pour effectuer une action dans un formulaire sans savoir quel objet est actuellement sélectionné. N'oubliez pas auparavant de tester si l'objet est du type voulu, à l'aide de la fonction Type.

Note : Lorsqu'elle est utilisée avec une list box, la fonction Objet focus retourne un pointeur vers la list box ou la colonne de la list box en fonction du contexte. Pour plus d'informations, reportez-vous à la section Gestion programmée des objets de type List box.

Cette commande ne peut pas être utilisée sur les champs dans les sous-formulaires.

Note : Cette commande n'a de sens qu'en cours de saisie. Son utilisation hors de ce contexte génère des messages d'erreur.

Exemple

L'exemple suivant est une méthode objet pour un bouton. Cette méthode passe les données de l'objet courant en majuscules. L'objet doit être de type Texte ou Alpha (type 0 ou 24) :

```
$pointeur := Objet focus  ` Obtenir le pointeur vers le dernier objet
Au cas ou
   :(Nil($pointeur))  ` Aucun objet n'a le focus
      ...
   :((Type ($pointeur->) = Est un champ alpha) | (Type($pointeur->) =
                                                   Est une variable chaîne))
         ` S'il s'agit d'un objet de type Texte ou Alpha
      $pointeur-> := Majusc ($pointeur->)  ` Mettre les données en majuscules
Fin de cas
```

Pop up menu (contenu{; parDéfaut{; coordX; coordY}}) → Numérique

Paramètre	Type		Description
contenu	Texte	→	Définition du texte du menu
parDéfaut	Numérique	→	Numéro de l'élément sélectionné par défaut
coordX	Numérique	→	Coordonnée X du coin supérieur gauche
coordY	Numérique	→	Coordonnée Y du coin supérieur gauche
Résultat	Numérique	←	Numéro de l'élément de menu sélectionné

Description

La commande Pop up menu fait apparaître un pop up à l'emplacement courant du curseur de la souris ou à l'emplacement défini par les paramètres facultatifs coordX et coordY.

Selon les règles standard d'interface utilisateur, cette commande doit généralement être appelée en réponse à un clic souris, et lorsque le bouton reste enfoncé un certain laps de temps.

Vous définissez les éléments du pop up menu à l'aide du paramètre contenu, de la manière suivante :
• Chaque élément est séparé des autres par un point-virgule (;), "Elément1;Elément2;Elément3".
• Pour inactiver un élément, placez une parenthèse ouvrante "(" dans son libellé.
• Pour définir une ligne de séparation, passez la valeur "(-" en tant que libellé.
• Pour définir le style de caractères d'un élément, placez dans son libellé le symbole inférieur à "<" suivi d'une lettre. Voici les différents codes :

<B	Gras
<I	Italique
<U	Souligné
<O	Contours (Mac OS seulement)
<S	Relief (Mac OS seulement)

• Pour associer une coche à un élément, insérez dans son libellé un point d'exclamation "!" suivi du caractère que vous voulez utiliser comme coche.
- Sous Mac OS, le caractère passé est directement affiché. Pour afficher la coche standard quelle que soit la version ou la langue du système, utilisez l'instruction Caractere(18).

- Sous Windows, une coche standard est affichée (quel que soit le caractère passé).
• Pour associer une icône à un élément, insérez dans son libellé un accent circonflexe "^" suivi d'un caractère dont le code moins 48 plus 256 (ou plus 208) représente un numéro de ressource d'icône Mac OS.
• Pour ajouter un raccourci clavier à un élément, insérez dans son libellé une barre oblique "/" suivie du caractère de raccourci. Notez que cette dernière option est uniquement informative (aucun raccourci clavier n'active le pop up menu), cependant vous pouvez indiquer un raccourci clavier si l'élément du pop up menu dispose d'une commande équivalente dans la barre de menus principale de votre application.

Astuce : Il est possible de désactiver le mécanisme d'interprétation des caractères spéciaux (!, /, etc.) dans le pop up menu afin, par exemple, de faire figurer ces caractères dans les libellés. Pour cela, il suffit de faire débuter le paramètre contenu par l'instruction Caractere(1) puis d'utiliser cette instruction comme séparateur :

```
contenu:=Caractere(1)+"1/4"+Caractere(1)+"1/2"+Caractere(1)+"3/4")
```

A noter qu'une fois cette instruction exécutée, il n'est plus possible d'affecter de style ou de raccourcis au pop up menu.

Le paramètre optionnel parDéfaut vous permet de définir l'élément du pop up menu sélectionné par défaut lorsque celui-ci apparaît. Passez une valeur située entre 1 et le nombre d'éléments du menu. Si vous ne passez pas ce paramètre, le premier élément du menu sera sélectionné par défaut. Si vous passez également les paramètres coordX et coordY (cf. ci-dessous), ce paramètre est ignoré.

Les paramètres facultatifs coordX et coordY permettent de désigner l'emplacement du pop up menu à afficher. Passez respectivement dans coordX et coordY les coordonnées horizontale et verticale du coin supérieur gauche du menu. Ces coordonnées doivent être exprimées en pixels dans le système de coordonnées local au formulaire courant. Ces deux paramètres doivent être passés ensemble ; si un seul est passé, il est ignoré.
Si vous utilisez les paramètres coordX et coordY, le paramètre parDéfaut est ignoré. Dans ce cas en effet, la souris ne se trouve pas nécessairement au niveau du pop up menu.
Ces paramètres sont utiles notamment pour la gestion des boutons 3D avec pop up menu associé.

Lorsqu'un élément du pop up menu est sélectionné, la commande retourne son numéro, autrement elle retourne zéro.

Note : Utilisez les pop up menus avec un nombre "raisonnable" d'éléments. Si, par exemple, vous voulez afficher plus de 50 éléments, envisagez plutôt d'employer une zone de défilement dans un formulaire.

Exemple

La méthode projet MON RACCOURCI fait apparaître un pop up menu de navigation :

```
    ` Méthode projet MON RACCOURCI
POSITION SOURIS($vlMouseX;$vlMouseY;$vlBouton)
Si (Macintosh control enfoncee | ($vlBouton=2))
    $vtItems:="A propos de cette base...<l;(-;!-Autres options;(-"
    Boucle ($vlTable;1;Lire numero derniere table)
        Si(Est un numero de table valide($vlTable))
            $vtItems:=$vtItems+";"+Nom de la table($vlTable)
        Fin de si
    Fin de boucle
    $vlChoixUtilisateur:=Pop up menu($vtItems)
    Au cas ou
        : ($vlChoixUtilisateur=1)
            ` Afficher les informations
        : ($vlChoixUtilisateur=2)
            ` Afficher les options
        Sinon
            Si ($vlChoixUtilisateur>0)
                ` Aller à la table dont le numéro est $vlChoixUtilisateur-4
            Fin de si
    Fin de cas
Fin de si
```

Cette méthode projet peut être appelée d'une des manières suivantes :
• depuis la méthode d'un objet réagissant à un clic souris, et n'attendant pas que le bouton soit relâché (par exemple un bouton invisible),
• depuis un process qui “épie” les événements et communique avec les autres process,
• depuis une méthode de gestion d'événements installée par la commande APPELER SUR EVENEMENT.

Dans les deux derniers cas, il n'est pas nécessaire que le clic se produise dans un objet de formulaire. C'est l'un des avantages de la commande Pop up menu. Généralement, les pop up menus sont affichés par l'intermédiaire d'objets de formulaire. Avec Pop up menu, vous pouvez faire apparaître un pop up menu n'importe où.

Le pop up menu s'affiche sous Windows lorsque l'utilisateur appuie sur le **bouton droit** de la souris, et sous Mac OS lorsqu'il utilise la combinaison **Control+clic**. Notez cependant que la méthode ci-dessus ne teste pas le clic souris, c'est la méthode appelante qui en est chargée.

Voici le pop up menu tel qu'il s'affiche sous Windows (à gauche) et sous Mac OS (à droite). Notez la coche standard de la version Windows :

Référence

Pop up menu dynamique, POSITION SOURIS.

POSITION SOURIS (sourisX; sourisY; boutonSouris{; *})

Paramètre	Type		Description
sourisX	Numérique	←	Coordonnée horizontale de la souris
sourisY	Numérique	←	Coordonnée verticale de la souris
boutonSouris	Numérique	←	Etat du bouton de la souris : 0 = Bouton relâché 1 = Bouton enfoncé 2 = Bouton droit enfoncé 3 = Les deux boutons enfoncés
*		→	Si spécifié, utiliser le système de coordonnées globales Si omis, utiliser le système de coordonnées locales

Description

La commande POSITION SOURIS retourne l'état courant de la souris.

Les coordonnées horizontale et verticale sont retournées dans les paramètres sourisX et sourisY. Si vous passez le paramètre *, ces coordonnées sont exprimées par rapport à l'écran. Si vous ne passez pas le paramètre *, vous exprimez ces coordonnées par rapport à la fenêtre de premier plan du process courant.

Le paramètre boutonSouris retourne l'état du ou des bouton(s) de la souris, comme décrit ci-dessus dans le tableau des paramètres.

Note : Les valeurs 2 et 3 peuvent être retournées sous Mac OS X à compter de la version 10.2.5 uniquement.

Exemple

Reportez-vous à l'exemple de la commande Pop up menu.

Référence

APPELER SUR EVENEMENT, Macintosh commande enfoncee, Macintosh control enfoncee, Macintosh option enfoncee, Majuscule enfoncee, Verrouillage majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

REDESSINER (objet)

Paramètre	Type		Description
objet	Objet	→	Table de laquelle redessiner le sous-formulaire ou Champ duquel redessiner la zone ou Variable de laquelle redessiner la zone ou Table du formulaire à actualiser sur les navigateurs

Description

Lorsque vous modifiez par programmation le contenu d'un champ affiché dans un sous-formulaire, vous devez exécuter la commande REDESSINER pour vous assurer que le formulaire est correctement mis à jour.

Note Serveur Web : Lorsqu'elle est associée à l'événement formulaire Sur minuteur, la commande REDESSINER permet de provoquer la mise à jour d'un formulaire 4D affiché par un navigateur Web. Pour plus d'informations sur ce point, reportez-vous à la description de la commande FIXER MINUTEUR.

Référence

FIXER MINUTEUR.

SELECTIONNER TEXTE (zone; débutSél; finSél)

Paramètre	Type		Description
zone	Champ | Variable	→	Champ ou variable saisissable
débutSél	Numérique	→	Nouvelle position de début de sélection de texte
finSél	Numérique	→	Nouvelle position de fin de sélection de texte

Description

La commande SELECTIONNER TEXTE sélectionne une partie du texte dans zone.

Si zone n'est pas l'objet en cours de modification, elle récupère le focus.

Note : Cette commande ne peut pas être utilisée avec des champs situés dans le formulaire en mode liste d'un sous-formulaire.

Le paramètre débutSél représente la position du premier caractère à sélectionner, et le paramètre finSél représente la position du dernier caractère à sélectionner plus un. Si débutSél et finSél sont identiques, le point d'insertion est placé devant le caractère spécifié par débutSél et aucun caractère n'est sélectionné.

Si finSél est supérieur au nombre de caractères présents dans la zone, tous les caractères compris entre débutSél et la fin du texte sont sélectionnés.

Exemples

(1) L'exemple suivant sélectionne tous les caractères dans le champ saisissable [Produits]Notes :

```
SELECTIONNER TEXTE([Produits]Notes;1;Longueur([Produits]Notes)+1)
```

(2) L'exemple suivant place le point d'insertion au début du champ [Produits]Notes :

```
SELECTIONNER TEXTE([Produits]Notes;1;1)
```

(3) L'exemple suivant place le point d'insertion à la fin du champ [Produits]Notes :

```
$vLen:=Longueur([Produits]Notes)+1
SELECTIONNER TEXTE([Produits]Notes;$vLen;$vLen)
```

(4) Reportez-vous à l'exemple de la commande FILTRER FRAPPE CLAVIER.

Référence

TEXTE SELECTIONNE.

TEXTE SELECTIONNE (zone; débutSél; finSél)

Paramètre	Type		Description
zone	Champ | Variable	→	Champ ou variable saisissable
débutSél	Numérique	←	Position du début de la sélection de texte
finSél	Numérique	←	Position de la fin de la sélection de texte

Description

La commande TEXTE SELECTIONNE vous permet de déterminer précisément le texte actuellement sélectionné.

Attention : Bien que vous deviez passer à TEXTE SELECTIONNE un nom de variable ou de champ saisissable, cette commande ne retourne une position de sélection significative que lorsqu'elle est appliquée à la zone en cours de modification.

Note : Cette commande ne peut pas être utilisée avec des champs situés dans le formulaire en mode liste d'un sous-formulaire.

Le texte peut être sélectionné par l'utilisateur ou par la commande SELECTIONNER TEXTE.

Le paramètre débutSél retourne la position du premier caractère sélectionné.
Le paramètre finSél retourne la position du dernier caractère sélectionné plus un.

Si les valeurs débutSél et finSél retournées sont identiques, l'utilisateur n'a pas sélectionné de texte et le point d'insertion est placé devant le caractère spécifié par débutSél.

Exemples

(1) L'exemple suivant récupère le texte sélectionné dans le champ [Produits]Notes :

```
TEXTE SELECTIONNE ([Produits]Notes;vPremier;vDernier)
Si (vPremier<vDernier)
   ALERTE("Le texte sélectionné est : "+Sous chaine([Produits]Notes;vPremier;
                                                    vDernier-vPremier))
Fin de si
```

(2) Reportez-vous à l'exemple de la commande FILTRER FRAPPE CLAVIER.

Référence

FILTRER FRAPPE CLAVIER, Frappe clavier, SELECTIONNER TEXTE.

Verrouillage majuscule enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Verrouillage Majuscule

Description

Verrouillage majuscule enfoncee retourne Vrai si la touche **Verrouillage Majuscule** est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande Majuscule enfoncee.

Référence

Macintosh commande enfoncee, Macintosh control enfoncee, Macintosh option enfoncee, Majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

Windows Alt enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Windows Alt ou Etat de la touche Macintosh Option

Description

Windows Alt enfoncee retourne Vrai si la touche **Alt** Windows est enfoncée.

Note : Lorsqu'elle est appelée sous Mac OS, Windows Alt enfoncee retourne Vrai si la touche Macintosh **Option** est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande Majuscule enfoncee.

Référence

Macintosh commande enfoncee, Macintosh control enfoncee, Macintosh option enfoncee, Majuscule enfoncee, Verrouillage majuscule enfoncee, Windows Ctrl enfoncee.

Windows Ctrl enfoncee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Etat de la touche Ctrl Windows ou Etat de la touche Commande Macintosh

Description

Windows Ctrl enfoncee retourne Vrai si la touche **Ctrl** Windows est enfoncée.

Note : Lorsqu'elle est appelée sous Mac OS, Windows Ctrl enfoncee retourne Vrai si la touche Macintosh **Commande** est enfoncée.

Exemple

Reportez-vous à l'exemple de la commande Majuscule enfoncee.

Référence

Macintosh commande enfoncee, Macintosh option enfoncee, Majuscule enfoncee, Verrouillage majuscule enfoncee, Windows Alt enfoncee, Windows Ctrl enfoncee.

33

Interruptions

APPELER SUR EVENEMENT (méthodeEvén{; nomProcess})

Paramètre	Type		Description
méthodeEvén	Alpha	→	Méthode d'événement à appeler ou Chaîne vide pour arrêter l'interception des événements
nomProcess	Alpha	→	Nom de process

Description

APPELER SUR EVENEMENT installe la méthode dont le nom est passé dans méthodeEvén comme méthode de gestion des événements.

Conseil : Cette commande nécessite un niveau de connaissances avancé en programmation. Généralement, vous n'avez pas besoin d'appeler APPELER SUR EVENEMENT pour traiter les événements. Lorsque vous utilisez des formulaires, 4D gère pour vous les événements et les retourne aux formulaires et objets appropriés.

Astuce : Les commandes telles que POSITION SOURIS, Majuscule enfoncee, etc., permettent de récupérer des informations sur les événements. Ces commandes, dans une certaine mesure, peuvent être appelées depuis les méthodes objet pour traiter les informations dont vous avez besoin. Elles peuvent ainsi vous épargner l'écriture d'un algorithme basé sur une structure du type APPELER SUR EVENEMENT.

La portée de cette commande est la session de travail. Par défaut, la méthode est exécutée dans un process local séparé. Vous ne pouvez avoir qu'une méthode de gestion d'événement à la fois. Pour désinstaller une méthode de gestion d'événement, appelez de nouveau APPELER SUR EVENEMENT et passez une chaîne vide dans méthodeEvén.

Comme la méthode de gestion d'événement tourne dans process séparé, méthodeEvén est toujours active, même si aucune méthode 4D n'est en cours d'exécution. Après l'installation, 4D appelle la méthode méthodeEvén dès qu'un événement survient. Un événement peut être un clic souris ou la frappe d'une touche.

Le paramètre optionnel nomProcess permet de donner un nom au process créé par APPELER SUR EVENEMENT. Si nomProcess commence par le symbole dollar ($), nomProcess est un process local, ce dont vous aurez généralement besoin. Si vous ne passez pas le paramètre nomProcess, 4D crée par défaut un process local nommé $Gestionnaire d'événement.

ATTENTION : Soyez prudent lors de l'écriture d'une méthode de gestion d'événement. N'appelez pas de commande générant un événement, sinon vous risquez de ne plus pouvoir sortir de la méthode. La combinaison de touches **Ctrl+Maj+Retour Arrière** (sous Windows) ou **Commande+Option+Maj+Control+Retour Arrière** (sous Mac OS) convertit le process d'événement en un process normal, ce qui signifie que la méthode ne reçoit plus systématiquement tous les événements qui surviennent. Cette combinaison vous permet de sortir d'une méthode de gestion d'événement devenue incontrôlable.

Dans la méthode de gestion d'événement, vous pouvez lire les variables système suivantes : MouseDown, KeyCode, Modifiers, MouseX, MouseY et MouseProc. Notez que ces variables sont des variables process. Leur portée est donc le process de gestion d'événements. Copiez-les dans des variables interprocess si vous souhaitez que leurs valeurs soient disponibles dans un autre process.

• La variable système MouseDown contient 1 s'il y a eu un clic souris, 0 sinon.

• La variable système KeyCode contient le code du caractère tapé au clavier, ou le code d'une touche de fonction. Référez-vous aux sections Codes Unicode et Codes ASCII qui liste les codes de caractères utilisés par 4D, ainsi qu'à la section Codes des touches de fonction. 4D fournit des constantes prédéfinies pour les principaux codes ASCII et touches de fonctions. Vous pouvez les visualiser à l'aide la fenêtre de l'Explorateur, dans les thèmes correspondants.

• La variable système Modifiers permet de savoir si les touches suivantes étaient enfoncées lorsqu'un événement s'est produit :

Plate-forme	**Modifiers**
Windows	Maj, Verrouillage des majuscules, Alt, Ctrl, Bouton droit de la souris
Macintosh	Maj, Verrouillage des majuscules, Option, Contrôle, Commande

Notes
- La touche Windows Alt est l'équivalent de la touche Macintosh Option.
- La touche Windows Ctrl est l'équivalent de la touche Macintosh Commande.
- La touche Macintosh Control n'a pas d'équivalent sous Windows. Cependant, un clic bouton droit de la souris sous Windows est l'équivalent de Control+clic sur Macintosh.

Isolément, les touches "modifiers" ne génèrent pas d'événement. Une autre touche ou le bouton de la souris doit également être enfoncé(e). La variable Modifiers est une variable de type Entier long (4 octets), qui doit être considérée comme un tableau de 32 bits. 4D fournit des constantes prédéfinies exprimant la position ou le masque des bits pour tester le bit correspondant à chaque touche de modification. Lorsque, par exemple, vous voulez détecter si la touche Majuscule a été enfoncée pour l'événement, vous pouvez écrire :

```
Si (Modifiers ?? Bit touche majuscule)  ` Si la touche Majuscule était enfoncée
```

ou :

```
Si ((Modifiers & Masque touche majuscule)#0) ` Si la touche Majuscule était enfoncée
```

Note : Sous Windows, la valeur 128 est ajoutée à la variable Modifiers si le bouton (gauche) de la souris est relâché au moment de l'événement.

• Les variables systèmes MouseX et MouseY contiennent les coordonnées horizontale et verticale du clic souris, exprimées dans le système de coordonnées locales de la fenêtre dans laquelle le clic s'est produit. L'angle supérieur gauche de la fenêtre a les coordonnées 0,0. Ces variables n'ont de signification que lorsqu'un clic souris a eu lieu.

• La variable système MouseProc contient le numéro de référence du process dans lequel le clic souris s'est produit.

Note : Les variables système MouseDown, KeyCode, Modifiers, MouseX, MouseY et MouseProc ne contiennent des valeurs significatives que dans une méthode de gestion d'événement installée par APPELER SUR EVENEMENT.

Exemple

L'exemple suivant annule l'impression si l'utilisateur appuie sur les touches **Ctrl+.** (**Commande+.** sous Mac OS). En premier lieu, la méthode de gestion des événements est installée. Ensuite, un message s'affiche, indiquant que l'impression a été annulée. Si la variable interprocess <>vbOnStoppe est égale à Vrai dans la méthode de gestion d'événement, une boîte de dialogue d'alerte s'affiche pour indiquer à l'utilisateur le nombre d'enregistrements qui viennent de s'imprimer. Enfin, la méthode de gestion d'événement est désinstallée :

```
UTILISER PARAMETRES IMPRESSION
Si (OK =1)
   <>vbOnStoppe:=Faux
   APPELER SUR EVENEMENT("GESTION EVENEMENTS")
   TOUT SELECTIONNER([Personnes])
   MESSAGE("Pour interrompre l'impression, appuyez sur Ctrl+point.")
      $NbEnregistrements:=Enregistrements trouves([Personnes])
```

```
        Boucle ($Enrg;1;$NbEnregistrements)
            Si (<>vbOnStoppe)
                ALERTE("L'impression a été annulée à l'enregistrement "+Chaine($Enrg)+
                                                        " sur "+Chaine($NbEnregistrements))
                $Enrg:=$NbEnregistrements+1
            Sinon
                Imprimer ligne([Personnes];"Etat")
            Fin de si
        Fin de boucle
        SAUT DE PAGE
        APPELER SUR EVENEMENT("") ` Désinstallation de la méthode d'appel sur événement
    Fin de si
```

La méthode de gestion d'événement teste si la combinaison de touches **Ctrl+.** (**Commande+.**) a été employée et met la variable interprocess <>vbOnStoppe à Vrai :

```
` Méthode projet GESTION EVENEMENTS
Si ((Modifiers ?? Bit touche commande) & (KeyCode = Point))
    CONFIRMER("Voulez-vous vraiment annuler l'impression ?")
    Si (OK=1)
        <>vbOnStoppe:=Vrai
        FILTRER EVENEMENT
            ` N'oubliez pas cet appel sinon 4D traitera aussi cet événement
    Fin de si
Fin de si
```

Notez que APPELER SUR EVENEMENT est utilisé dans cet exemple car un état spécial est imprimé à l'aide des commandes PARAMETRES IMPRESSION, Imprimer ligne et SAUT DE PAGE dans une structure de type Boucle...Fin de boucle.

Lorsque vous imprimez un état à l'aide la commande IMPRIMER SELECTION, vous n'avez pas besoin de gérer les événements permettant à l'utilisateur d'interrompre l'impression, IMPRIMER SELECTION le fait pour vous.

Référence

FILTRER EVENEMENT, Majuscule enfoncee, Methode appelee sur evenement, POSITION SOURIS.

APPELER SUR ERREUR (méthodErreur)

Paramètre	Type		Description
méthodErreur	Alpha	→	Méthode de gestion d'erreur à appeler ou Chaîne vide pour désinstaller la méthode

Description

APPELER SUR ERREUR installe la méthode projet dont le nom est passé dans méthodErreur comme méthode d'interception des erreurs — aussi appelée méthode de gestion des erreurs.

La portée de cette commande est le process courant. Il ne peut y avoir qu'une seule méthode de gestion des erreurs par process, mais il peut exister différentes méthodes de gestions d'erreurs pour plusieurs process.

Pour désinstaller une méthode de gestion des erreurs, appelez de nouveau APPELER SUR ERREUR et passez une chaîne vide dans méthodErreur.

Après l'installation, 4D appelle cette méthode lorsqu'une erreur se produit.

Vous pouvez identifier les erreurs en lisant la variable système Error, qui contient le code de l'erreur. Les codes d'erreurs retournés par 4D sont traités dans les sections Codes d'erreurs. Reportez-vous par exemple à la section Erreurs de syntaxe ou Erreurs de la base de données. La variable Error n'est définie qu'à l'intérieur de la méthode de gestion des erreurs ; si vous souhaitez que le code soit accessible dans la méthode ayant provoqué l'erreur, copiez la variable Error dans votre propre variable process.

La méthode de gestion des erreurs doit généralement traiter les erreurs de manière appropriée ou afficher un message d'erreur à l'utilisateur. Les erreurs peuvent être générées par :
• Le moteur de base de données de 4D ; par exemple, lorsque la sauvegarde d'un enregistrement provoquerait la duplication d'une clé d'index unique.
• L'environnement de 4D ; par exemple, lorsque vous n'avez pas assez de mémoire pour remplir un tableau.
• Le système d'exploitation sur lequel la base est lancée ; par exemple, disque plein ou erreurs d'entrée/sortie.

La commande STOP peut être utilisée pour stopper le traitement. Si vous n'appelez pas STOP dans la méthode installée, 4D retourne à la méthode interrompue et reprend son exécution. Utilisez la commande STOP lorsque l'exécution ne peut se poursuivre.

Si une erreur se produit dans la méthode de gestion d'erreurs elle-même, 4D reprend le contrôle de la gestion des erreurs. En conséquence, assurez-vous que la méthode de gestion des erreurs installée ne puisse pas elle-même générer d'erreur. Aussi, vous ne pouvez pas utiliser la commande APPELER SUR ERREUR dans une méthode de gestion des erreurs.

APPELER SUR ERREUR est généralement placée dans la méthode base d'ouverture d'une application, afin de gérer les erreurs pour cette application. APPELER SUR ERREUR peut également être placée au début d'une méthode pour gérer les erreurs spécifiques à cette méthode.

Lorsqu'une méthode APPELER SUR ERREUR est installée, il n'est plus possible de tracer l'exécution des méthodes à l'aide de la combinaison **Alt+clic** (sous Windows) ou **Option+clic** (sous Mac OS). En effet, cette combinaison génère un code d'erreur (code 1006) qui active immédiatement la méthode d'appel sur erreur. Cependant, vous pouvez tester ce code d'erreur et appeler la commande TRACE si nécessaire.

Exemples

(1) La méthode projet suivante tente de créer un document dont le nom est reçu en paramètre et retourne 0 (zéro) ou un code d'erreur si le document n'a pas pu être créé :

```
      ` Méthode projet Créer doc
      ` Créer doc ( Chaîne ; Pointeur ) -> Entier long
      ` Créer doc ( NomDoc ; ->DocRef ) -> Code d'erreur résultant

gError:=0
APPELER SUR ERREUR("IO TRAITEMENT ERREURS")
$2->:=Creer document($1)
APPELER SUR ERREUR("")
$0:=gError
```

La méthode projet IO TRAITEMENT ERREURS est la suivante :

```
      ` Méthode projet IO TRAITEMENT ERREURS
gError:=Error ` Simple copie du code d'erreur dans la variable process gError
```

Notez l'utilisation de la variable process gError pour récupérer le code d'erreur dans la méthode en train de s'exécuter. Une fois que ces méthodes sont présentes dans votre base, vous pouvez écrire par exemple :

```
    ` ...
C_HEURE(vhDocRef)
$vlErrCode:=Créer doc($vsDocumentNom;->vhDocRef)
Si ($vlErrCode=0)
        `...
    FERMER DOCUMENT($vlErrCode)
Sinon
    ALERTE ("Le document n'a pas pu être créé, erreur d'E/S "+Chaîne($vlErrCode))
Fin de si
```

(2) Reportez-vous à l'exemple de la section Tableaux et mémoire.

(3) Alors que vous implémentez un ensemble complexe d'opérations, vous pouvez terminer avec de multiples sous-routines qui nécessitent différentes méthodes de gestion des erreurs. Comme ne pouvez avoir qu'une seule méthode à la fois de gestion des erreurs par process, vous devez soit repérer la méthode courante à chaque fois que vous appelez APPELER SUR ERREUR, soit utiliser une variable tableau process (ici tabErrorMethod) pour “empiler” les méthodes de gestion d'erreur ainsi qu'une méthode projet (ici APPEL SUR ERR) pour les installer et les désinstaller. Le tableau doit être initialisé au tout début de l'exécution du process :

```
    ` N'oubliez pas d'initialiser le tableau au début
    ` de la méthode de gestion du process
TABLEAU ALPHA(63;tabErrorMethod;0)
```

Voici la méthode personnalisée APPEL SUR ERR :

```
    ` Méthode projet APPEL SUR ERR
    ` APPEL SUR ERR { ( Chaîne ) }
    ` APPEL SUR ERR { ( Nom de la méthode ) }

C_ALPHA(63;$1;$ErrorMethod)
C_ENTIER LONG($vlElem)

Si (Nombre de parametres>0)
    $ErrorMethod:=$1
Sinon
    $ErrorMethod:=""
                        Fin de si
```

```
Si ($ErrorMethod#"")
    C_ENTIER LONG(gError)
    gError:=0
    $vlElem:=1+Taille tableau(tabErrorMethod)
    INSERER DANS TABLEAU(tabErrorMethod;$vlElem)
    tabErrorMethod{$vlElem}:=$1
    APPELER SUR ERREUR($1)
Sinon
    APPELER SUR ERREUR("")
    $vlElem:=Taille tableau(tabErrorMethod)
    Si ($vlElem>0)
        SUPPRIMER DANS TABLEAU(tabErrorMethod;$vlElem)
        Si ($vlElem>1)
            APPELER SUR ERREUR(tabErrorMethod{$vlElem-1})
        Fin de si
    Fin de si
Fin de si
```

Vous pouvez alors l'appeler de la manière suivante :

```
gError:=0
APPEL SUR ERR("ERREURS ES") ` Installe la méthode de gestion d'erreurs ERREURS ES
    ` ...
APPEL SUR ERR("TOUTES ERREURS")
    ` Installe la méthode de gestion d'erreurs TOUTES ERREURS
    ` ...
APPEL SUR ERR
    ` Désinstalle la méthode de gestion d'erreurs TOUTES ERREURS et réinstalle ERREURS ES
    ` ...
APPEL SUR ERR   ` Désinstalle la méthode de gestion d'erreurs ERREURS ES
    ` ...
```

(4) La méthode de gestion d'erreurs suivante ignore les interruptions de l'utilisateur :

```
    ` Méthode projet MONTRER ERREURS SEULEMENT
Si (Error#1006)
    ALERTE ("L'erreur "+Chaine(Error)+" s'est produite.")
Fin de si
```

Référence

Methode appelee sur erreur, STOP.

FILTRER EVENEMENT

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

FILTRER EVENEMENT doit être appelée à l'intérieur d'une méthode de gestion d'événements installée par APPELER SUR EVENEMENT.

Lorsqu'une méthode de gestion d'événements appelle la commande FILTRER EVENEMENT, l'événement courant n'est pas passé à 4D.

Cette commande vous permet d'effacer l'événement courant (i.e. clic, frappe clavier) de la séquence d'événements, de manière à ce que 4D n'effectue pas de traitement sur l'événement que vous provoquez dans la méthode de gestion d'événements.

ATTENTION : Evitez de créer une méthode de gestion d'événement appelant uniquement FILTRER EVENEMENT car TOUS les événements vont être ignorés par 4D. Si vous vous retrouvez dans un tel cas, vous pouvez sortir de la méthode en tapant **Ctrl**+**Maj**+**Retour Arrière** (sous Windows) ou **Commande**+**Option**+**Maj**+**Contrôle**+**Retour Arrière** (sous Mac OS). Dans ce cas, le process de gestion d'événement est converti en process normal n'interceptant plus aucun événement.

Cas particulier : La commande FILTRER EVENEMENT peut également être utilisée au sein d'une méthode de formulaire sortie standard, lorsque le formulaire est affiché par l'intermédiaire des commandes VISUALISER SELECTION ou MODIFIER SELECTION. Dans ce cas précis, la commande FILTRER EVENEMENT permet de filtrer les double-clics sur les enregistrements (et ainsi, exécuter d'autres actions que l'ouverture des enregistrements en mode page).
Pour cela, il vous suffit de placer dans la méthode du formulaire sortie les lignes suivantes :

```
Si(Evenement formulaire=Sur double clic souris)
   FILTRER EVENEMENT
   ...  `Traiter le double-clic
Fin de si
```

Exemple

Référez-vous à l'exemple d'APPELER SUR EVENEMENT.

Référence

APPELER SUR EVENEMENT.

LIRE PILE DERNIERE ERREUR (tabCodes; tabCompInternes; tabLibellés)

Paramètre	Type		Description
tabCodes	Tab numérique	←	Tableau de numéros d'erreurs
tabCompInternes	Tab chaîne	←	Tableau de codes de composants internes
tabLibellés	Tab chaîne	←	Tableau de libellés d'erreurs

Description

La commande LIRE PILE DERNIERE ERREUR retourne les informations relatives à la “pile” d'erreurs courante de l’application 4D. Lorsqu’une instruction 4D provoque une erreur, la pile d’erreurs courante contient la description de l’erreur ainsi que les éventuelles erreurs générées en cascade. Par exemple l’erreur du type "disque saturé" entraîne une erreur d'écriture dans le fichier puis une erreur dans la commande de sauvegarde d’enregistrements : la pile contient alors trois erreurs. Si la dernière instruction 4D n’a pas généré d’erreur, la pile d’erreurs courante est vide.

Cette commande générique permet de traiter tous les types d’erreurs pouvant se produire dans l'application 4D.

Note : Toutefois, pour obtenir des informations détaillées relatives aux erreurs générées par une source ODBC, il est nécessaire d’utiliser la commande SQL LIRE DERNIERE ERREUR.

La commande LIRE PILE DERNIERE ERREUR doit être appelée dans le cadre d’une méthode d’appel sur erreur installée par la commande APPELER SUR ERREUR.

Les informations sont retournées sous la forme de trois tableaux synchronisés :
• tabCodes : ce tableau reçoit la liste des codes d’erreurs générés.
• tabCompInternes : ce tableau contient les codes des composants internes associés à chaque erreur.
• tabLibellés : ce tableau contient les libellés de chaque erreur.

La liste des codes d’erreurs et de leurs libellés est fournie dans les sections du thème "Code d'erreurs".

Référence

APPELER SUR ERREUR, SQL LIRE DERNIERE ERREUR.

Methode appelee sur erreur → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom de la méthode d'appel sur erreur

Description

La commande Methode appelee sur erreur retourne le nom de la méthode installée par la commande APPELER SUR ERREUR pour le process courant.

Si aucune méthode d'appel sur erreur n'a été installée, une chaîne vide ("") est retournée.

Exemple

Cette commande est particulièrement utile dans le cadre des composants, car elle permet de changer temporairement puis de rétablir les méthodes d'interception d'erreurs :

```
$methCourante:=Methode appelee sur erreur
APPELER SUR ERREUR("NouvelleMéthode")
    ` Si le document ne peut être ouvert, une erreur est générée
$ref:=Ouvrir document("MonDocument")
    ` Réinstallation de la méthode précédente
APPELER SUR ERREUR($methCourante)
```

Référence

APPELER SUR ERREUR.

Methode appelee sur evenement

Methode appelee sur evenement → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom de la méthode d'appel sur evenement

Description

La commande Methode appelee sur evenement retourne le nom de la méthode installée par la commande APPELER SUR EVENEMENT.

Si aucune méthode d'appel sur événement n'a été installée, une chaîne vide ("") est retournée.

Référence

APPELER SUR EVENEMENT.

Note : Vous aurez rarement besoin de cette commande.

STOP

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande STOP est destinée à être utilisée dans une méthode projet de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Si vous n'avez pas installé de méthode projet de gestion d'erreurs, lorsqu'une erreur se produit (par exemple une erreur de la base de données), 4D affiche sa boîte de dialogue d'erreur standard et interrompt l'exécution de votre code :
• Si le code en cours d'exécution est une méthode objet ou formulaire (ou une méthode projet appelée depuis une méthode formulaire ou objet), 4D "rend la main" au formulaire actuellement affiché.
• Si le code en cours d'exécution est une méthode appelée depuis un menu, 4D "rend la main" à la barre de menus ou au formulaire actuellement affiché.
• Si le code en cours d'exécution est la méthode de gestion d'un process, le process est tué.
• Si le code en cours d'exécution est une méthode appelée directement ou indirectement par une opération d'import ou d'export, cette dernière est stoppée. Il en va de même pour les recherches séquentielles et les tris.
• Etc...

Si vous décidez de traiter les erreurs à l'aide d'une méthode projet d'interception d'erreurs, 4D n'affiche plus sa boîte de dialogue d'erreur standard et n'interrompt plus l'exécution de votre code. Au lieu de cela, 4D appelle votre méthode projet d'interception d'erreurs puis poursuit l'exécution de la ligne de code suivant celle ayant provoqué l'erreur. Vous pouvez traiter certaines erreurs par programmation (par exemple pendant un import, si vous interceptez une erreur de la base de donnée signalant une valeur dupliquée, vous pouvez ignorer l'erreur et poursuivre l'opération). Il existe également des erreurs que vous ne pouvez pas traiter ou des erreurs que vous ne devez pas "ignorer". Dans ces cas, vous devez stopper l'exécution de la méthode comme le fait 4D ; pour cela, appelez la commande STOP depuis la méthode projet d'interception d'erreurs.

Un peu d'histoire...
Bien que la commande STOP soit destinée à une utilisation au sein d'une méthode projet d'interception d'erreurs, des membres de la communauté 4D ont commencé à l'utiliser dans d'autres méthodes projet pour interrompre leur exécution. Le fait que cela fonctionne n'est qu'un "effet secondaire". Nous vous recommandons de n'utiliser cette commande que dans des méthodes projet d'interception d'erreurs.

Référence

APPELER SUR ERREUR.

34

Langage

Est une variable

Est une variable (unPointeur) → Booléen

Paramètre	Type		Description
unPointeur	Pointeur	→	Pointeur à tester
Résultat	Booléen	←	VRAI = Pointeur pointe vers une variable FAUX = Pointeur ne pointe pas vers une variable

Description

La fonction Est une variable retourne Vrai si le pointeur passé dans le paramètre unPointeur référence une variable définie. Elle retourne Faux dans tous les autres cas (pointeur vers un champ ou table, pointeur Nil, etc.).

A partir de la version 6, vous pouvez utiliser la commande RESOUDRE POINTEUR qui vous indique la nature de l'objet référencé, quel qu'il soit (pointeurs Nil compris).

Référence

Nil, RESOUDRE POINTEUR.

EXECUTER METHODE (nomMéthode{; résultat | *{; param}}{; param2; ...; paramN})

Paramètre	Type		Description
nomMéthode	Chaîne	→	Nom de méthode projet à exécuter
résultat | *	Variable | *	←	Variable recevant le résultat de la méthode ou * pour une méthode ne retournant pas de résultat
param	Expression	→	Paramètre(s) de la méthode

Description

La commande EXECUTER METHODE provoque l'exécution de la méthode projet nomMéthode en lui passant éventuellement les paramètres param1...paramN. Vous pouvez passer tout nom de méthode appelable depuis la base ou le composant exécutant la commande.

Passez dans résultat une variable devant recevoir le résultat de l'exécution de nomMéthode (valeur placée dans $0 à l'intérieur de nomMéthode). Si la méthode ne retourne pas de résultat, passez * comme deuxième paramètre. Si la méthode ne retourne pas de résultat et ne nécessite pas non plus le passage de paramètre(s), passez uniquement le paramètre nomMéthode.

Le contexte d'exécution est préservé dans la méthode appelée, ce qui signifie que le formulaire courant et l'éventuel événement formulaire courant restent définis.

Si vous appelez cette commande depuis un composant et passez dans nomMéthode un nom de méthode appartenant à la base hôte (ou inversement), la méthode doit avoir été partagée (option "Partager entre composants et base hôte" dans les propriétés de la méthode).

Référence

EXECUTER FORMULE.

Variables et ensembles système

Si cette commande est exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Nil (unPointeur) → Booléen

Paramètre	Type		Description
unPointeur	Pointeur	→	Pointeur à tester
Résultat	Booléen	←	VRAI = Pointeur Nil (->[]) FAUX = Pointeur valide vers un objet existant

Description

Nil retourne Vrai si le pointeur que vous passez dans unPointeur est Nil (->[]). Elle retourne Faux dans tous les autres cas (pointeur vers un champ, une table ou une variable).

A compter de la version 6 de 4D, vous pouvez utiliser la commande RESOUDRE POINTEUR qui vous indique le type de l'objet référencé, quel qu'il soit (pointeurs Nil compris).

Référence

Est une variable, RESOUDRE POINTEUR.

Nom commande (commande) → Alpha

Paramètre	Type		Description
commande	Numérique	→	Numéro de la commande
Résultat	Alpha	←	Nom (traduit) de la commande

Description

La fonction Nom commande retourne le nom (universel) de la commande dont le numéro a été passé dans commande.

4D comporte un système unique en son genre de traduction dynamique des mots-clés, constantes et noms de commandes que vous utilisez dans vos méthodes. Si, par exemple, vous écrivez dans la version anglaise de 4D :

```
DEFAULT TABLE ([MyTable])
ALL RECORDS ([MyTable])
```

Le même code, si vous l'ouvrez avec la version française de 4D, sera automatiquement traduit :

```
TABLE PAR DEFAUT ([MyTable])
TOUT SELECTIONNER ([MyTable])
```

Cependant, 4D comporte aussi une autre fonctionnalité unique, la commande EXECUTER FORMULE, qui vous permet de construire à la volée et d'exécuter des parties de code même lorsque la base de données est compilée.

Si nous réécrivons le code précédent avec des commandes EXECUTE FORMULA, en anglais, il prendra l'apparence suivante :

```
EXECUTE FORMULA ( "DEFAULT TABLE([MyTable])")
EXECUTE FORMULA ( "ALL RECORDS([MyTable])")
```

Le même code, ouvert avec la version française de 4D, sera automatiquement traduit :

```
EXECUTER FORMULE ( "DEFAULT TABLE([MyTable])")
EXECUTER FORMULE ( "ALL RECORDS([MyTable])")
```

4D traduit automatiquement la commande EXECUTE FORMULA (anglais) en EXECUTER FORMULE (français) mais ne peut pas traduire les instructions passées à la commande.

Si vous utilisez la commande EXECUTER FORMULE dans votre application et si vous souhaitez éliminer les problèmes de traduction liés à ce type d'instruction, utilisez Nom commande pour rendre vos instructions indépendantes des localisations. L'exemple précédent devient alors :

```
EXECUTE FORMULA (Command name (46)+"([MyTable])")
EXECUTE FORMULA (Command name (47)+"([MyTable])")
```

La version française se lira :

```
EXECUTER FORMULE (Nom commande (46)+"([MyTable])")
EXECUTER FORMULE (Nom commande (47)+"([MyTable])")
```

Note : Pour connaître le numéro de chaque commande, reportez-vous à la section Syntaxe des commandes (liste alphabétique).

Exemples

(1) Pour toutes les tables de votre base de données, vous avez créé un formulaire appelé "FORMULAIRE ENTREE" que vous utilisez pour la saisie dans chaque table. Vous voulez ajouter une méthode projet générique qui va désigner ce formulaire comme étant le formulaire entrée pour la table dont vous passez le nom ou le pointeur. Vous pouvez écrire :

```
    ` méthode projet FORMULAIRE ENTREE STANDARD
    ` FORMULAIRE ENTREE STANDARD ( Pointeur {; Chaîne })
    ` FORMULAIRE ENTREE STANDARD ( ->Table {; NomTable })
C_POINTEUR ($1)
C_ALPHA (31;$2)

Si (Nombre de parametres>=2)
    EXECUTER FORMULE (Nom commande (55)+"(["+$2+"];"FORMULAIRE ENTREE")")
Sinon
    Si (Nombre de parametres>=1)
        FORMULAIRE ENTREE ($1->;"FORMULAIRE ENTREE")
    Fin de si
Fin de si
```

Une fois que cette méthode a été ajoutée dans votre base, vous pouvez écrire :

```
FORMULAIRE ENTREE STANDARD  (->[Employés])
FORMULAIRE ENTREE STANDARD  ("Employés")
```

Note : Généralement, il est préférable d'utiliser des pointeurs pour écrire des routines génériques. Tout d'abord, le code sera exécuté compilé si la base est compilée. Ensuite, comme dans l'exemple ci-dessus, votre code cessera de fonctionner si vous renommez la table. Cependant, dans certains cas, l'utilisation de la commande EXECUTER FORMULE peut être la réponse à vos besoins.

(2) Dans un formulaire, vous voulez afficher une liste déroulante contenant les commandes standard de génération d'états. Dans la méthode objet de cette liste déroulante, vous écrivez :

```
Au cas ou
    : (Evenement formulaire =Sur Chargement)
        TABLEAU TEXTE (asCommand;4)
        asCommand{1}:=Nom commande (1) ` Somme
        asCommand{2}:=Nom commande (2) ` Moyenne
        asCommand{3}:=Nom commande (3) ` Min
        asCommand{4}:=Nom commande (4) ` Max
    ` ...
Fin de cas
```

Dans une version anglaise de 4D, la liste déroulante contiendra : Sum, Average, Min et Max.
Dans une version française de 4D, la liste déroulante contiendra : Somme, Moyenne, Min et Max.

Référence

EXECUTER FORMULE, Syntaxe des commandes (liste alphabétique).

Nom methode courante → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom de la méthode d'appel

Description

La commande Nom methode courante retourne le nom de la méthode dans laquelle elle est appelée. Cette commande est utile dans le cadre du débogage de méthodes génériques.

En fonction du type de méthode d'appel, la chaîne retournée peut prendre l'une des formes suivantes :

Méthode d'appel	Chaîne retournée
Méthode base	NomMéthode
Trigger	Trigger sur [NomTable]
Méthode projet	NomMéthode
Méthode formulaire	[NomTable]NomFormulaire
Méthode objet	[NomTable]NomFormulaire.NomObjet

Cette commande ne doit pas être appelée depuis une formule 4D.

Note : Pour que cette commande fonctionne en mode compilé, il est nécessaire que la base ait été compilée avec l'option **Contrôle d'exécution** (située dans les Préférences de l'application) activée.
Pour désactiver localement le contrôle d'exécution dans une méthode (ou une partie de méthode), vous pouvez utiliser les commentaires spéciaux suivants :

`%R- pour désactiver le contrôle d'exécution
`%R+ pour activer le contrôle d'exécution
`%R* pour restituer l'état initial du contrôle d'exécution (défini dans les Préférences).

Nombre de parametres → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Nombre de paramètres effectivement passés

Description

Nombre de parametres retourne le nombre de paramètres passés à une méthode projet.

ATTENTION : Nombre de parametres n'a d'intérêt que dans une méthode projet appelée par une autre méthode (projet ou non). Si la méthode projet qui appelle Nombre de parametres est associée à une commande de menu, la fonction retourne 0.

Exemples

(1) Les méthodes projet de 4D acceptent que des paramètres soient optionnels, à partir de la droite. Par exemple, la méthode maMéthode(a;b;c;d) peut accepter les syntaxes suivantes :

```
maMéthode ( a ; b ; c ; d )  ` Tous les paramètres sont passés
maMéthode ( a ; b ; c )  ` Le dernier paramètre n'est pas passé
maMéthode ( a ; b )  ` Les deux derniers paramètres ne sont pas passés
maMéthode ( a )  ` Seul le premier paramètre est passé
maMéthode  ` Aucun paramètre n'est passé
```

Si vous utilisez Nombre de parametres dans maMéthode, vous pouvez détecter le nombre de paramètres passés et effectuer des opérations différentes selon ce nombre. L'exemple suivant affiche un texte et peut soit l'insérer dans une zone de 4D Write, soit l'écrire dans un document sur disque :

```
` Méthode AJOUTER TEXTE
` AJOUTER TEXTE ( Texte { ; Entier long { ; Heure } } )
` AJOUTER TEXTE ( Texte { ; zone 4D Write { ; RéfDoc } } )

C_TEXTE ($1)
C_HEURE ($2)
C_ENTIER LONG ($3)
```

```
MESSAGE ($1)
Si (Nombre de parametres>=3)
    ENVOYER PAQUET ($3;$1)
Sinon
    Si (Nombre de parametres>=2)
        wr_INSERER TEXTE ($2;$1)
    Fin de si
Fin de si
```

Vous pouvez ensuite appeler cette méthode de ces trois façons différentes :

```
AJOUTER TEXTE (vtTexte)  ` Afficher seulement le message texte
AJOUTER TEXTE (vtTexte;$wrZone)  ` Afficher le message texte et ajouter le texte à $wrZone
AJOUTER TEXTE (vtTexte;0;$vhRéfDoc)  ` Afficher le message texte et l'écrire dans $vhRéfDoc
```

(2) Les méthodes projet de 4D acceptent un nombre variable de paramètres du même type à partir de la droite. Pour déclarer ces paramètres, vous devez utiliser des directives de compilation auxquelles vous passez ${N} en tant que variable, où N spécifie le premier des paramètres. A l'aide de Nombre de parametres, vous pouvez référencer ces paramètres dans une boucle avec la syntaxe d'indirection de paramètre. L'exemple suivant est une fonction qui retourne la valeur maximale reçue en tant que paramètre :

```
    ` Méthode projet Max de
    ` Max de ( Réel { ; Réel2... ; RéelN } ) -> Réel
    ` Max de ( Valeur { ; Valeur2... ; ValeurN } ) -> Valeur maximale
C_REEL ($0;${1})  ` Tous les paramètres sont de type REEL ainsi que le résultat de la fonction
$0:=${1}
Boucle ($vlParam;2;Nombre de parametres)
    Si (${$vlParam}>$0)
        $0:=${$vlParam}
    Fin de si
Fin de boucle
```

Vous pouvez alors appeler cette méthode d'une des deux manières suivantes :

```
vrRésultat:=Max de (Enregistrements dans ensemble("Opération A");
                                    Enregistrements dans ensemble("Opération B"))
```

ou :

```
vrRésultat:=Max de (r1;r2;r3;r4;r5;r6)
```

Référence

Commandes du thème Compilateur.

PAS DE TRACE

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande PAS DE TRACE est utilisée en phase de développement d'une base de données, pour contrôler l'exécution des méthodes.

PAS DE TRACE désactive le débogueur appelé par la commande TRACE, par une erreur ou par l'utilisateur. Utiliser PAS DE TRACE équivaut à cliquer sur le bouton **Pas de trace** dans la fenêtre de débogage.

Dans les bases compilées, cette commande est ignorée.

Référence

TRACE.

Pointeur vers (nomVar) → Pointeur

Paramètre	Type		Description
nomVar	Alpha	→	Nom d'une variable process ou interprocess
Résultat	Pointeur	←	Pointeur vers une variable process ou interprocess

Description

Pointeur vers retourne un pointeur vers la variable process ou interprocess dont le nom est passé dans nomVar.

Pour récupérer un pointeur vers un champ, utilisez la fonction Champ. Pour récupérer un pointeur vers une table, utilisez la fonction Table.

Note : Vous pouvez passer à Pointeur vers des expressions telles que, par exemple, tTabNom+"{3}". En revanche, vous ne pouvez pas utiliser d'éléments de tableau 2D (tTabNom+"{3}{5}") ni d'indices variables (tTabNom+"{maVar}").

Exemple

Dans un formulaire, vous construisez une grille de 5 X 10 variables saisissables dont les noms sont v1, v2... v50. Pour initialiser toutes ces variables, vous pouvez écrire :

```
` ...
Boucle ($vlVar;1;50)
   $vpVar:=Pointeur vers("v"+Chaine($vlVar))
   $vpVar->:=""
Fin de boucle
```

Référence

Champ, Table.

RESOUDRE POINTEUR (pointeur; nomVar; numTable; numChamp)

Paramètre	Type		Description
pointeur	Pointeur	→	Pointeur duquel récupérer l'objet référencé
nomVar	Alpha	←	Nom de la variable référencée ou chaîne vide
numTable	Numérique	←	Numéro de la table ou de l'élément de tableau référencé(e) ou 0 ou -1
numChamp	Numérique	←	Numéro du champ référencé ou 0

Description

RESOUDRE POINTEUR récupère l'information de l'objet référencé par pointeur et la retourne dans les paramètres nomVar, numTable et numChamp.

Selon la nature de l'objet référencé par le pointeur, RESOUDRE POINTEUR retourne les valeurs suivantes :

Objet référencé	Paramètres		
	nomVar	numTable	numChamp
Aucun (pointeur NIL)	"" (chaîne vide)	0	0
Variable	Nom de la variable	-1	0
Tableau	Nom du tableau	-1	0
Elément de tableau	Nom du tableau	numéro de l'élément	0
Table	"" (chaîne vide)	numéro de la table	0
Champ	"" (chaîne vide)	numéro de la table	numéro du champ

Notes :

• Si la valeur que vous passez dans le paramètre pointeur n'est pas de type pointeur, une erreur de syntaxe est générée.
• La commande RESOUDRE POINTEUR ne fonctionne pas avec les pointeurs vers des variables locales. En effet, par définition plusieurs variables locales de même nom pouvant exister à différents emplacements, il n'est pas possible pour la commande de connaître la variable à dépointer.

Exemples

(1) Dans un formulaire, vous créez un groupe de 100 variables saisissables qui s'appellent v1, v2... v100. Pour cela, vous procédez de la manière suivante :
• Vous créez une variable saisissable que vous appelez v.
• Vous définissez les propriétés de l'objet suivant vos besoins.
• Vous associez la méthode suivante à l'objet :

FaireQuelqueChose (**Self**) ` FaireQuelqueChose est une méthode projet de la base

• Vous pouvez alors soit dupliquer la variable autant de fois que nécessaire, soit utiliser la fonctionnalité Tableau sur la grille de l'éditeur de formulaires.
• Dans la méthode FaireQuelqueChose, si vous voulez connaître l'indice de la variable pour laquelle la méthode est appelée, vous écrivez le code suivant :

```
RESOUDRE POINTEUR($1;$vaNomVar;$vlNumTable;$vlNumChamp)
$vlVarNum:=Num(Sous chaine($vaNomVar;2))
```

• En suivant ces étapes, vous avez écrit une fois seulement les méthodes objet pour les 100 variables : vous n'avez pas eu besoin d'écrire FaireQuelqueChose(1), FaireQuelqueChose(2)..., FaireQuelqueChose(100)).

(2) Pour des raisons de débogage, vous voulez vérifier si le deuxième paramètre ($2) d'une méthode est un pointeur vers une table. Le début de votre méthode peut être écrit ainsi :

```
    ` ...
Si (<>Débogage)
    RESOUDRE POINTEUR($2;$vaNomVar;$vlNumTable;$vlNumChamp)
    Si (Non(($vlNumTable>0)&($vlNumChamp=0)&($vlNomVar="")))
        ` ATTENTION : Le pointeur n'est pas une référence à une table
        TRACE
    Fin de si
Fin de si
    ` ...
```

(3) Reportez-vous à l'exemple de la commande PROPRIETES GLISSER DEPOSER

Référence

Champ, Est une variable, Nil, Pointeur vers, PROPRIETES GLISSER DEPOSER, Table.

Self → Pointeur

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Pointeur	←	Pointeur vers l'objet du formulaire dont la méthode est en cours d'exécution (le cas échéant) Sinon Nil (->[]) si hors contexte

Description

Self retourne un pointeur vers l'objet du formulaire dont la méthode objet est en cours d'exécution.

La fonction Self est utilisée pour référencer une variable dans sa propre méthode objet. Elle retourne un pointeur valide si elle est appelée dans une méthode objet ou dans une méthode projet appelée directement ou indirectement par un méthode objet.

Si Self est appelée en-dehors de ce contexte, elle retourne un pointeur Nil (->[]).

Conseil : Self est très utile lorsque plusieurs objets d'un formulaire doivent effectuer la même action, opérée sur eux-mêmes.

Note : Lorsqu'elle est utilisée avec une list box, la fonction Self retourne un pointeur vers la list box ou la colonne de la list box en fonction du contexte. Pour plus d'informations, reportez-vous à la section Gestion programmée des objets de type List box.

Exemple

Référez-vous à l'exemple de la commande RESOUDRE POINTEUR.

Référence

RESOUDRE POINTEUR.

TRACE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande TRACE est utilisée, lors du développement des bases, pour tracer des méthodes, c'est-à-dire contrôler leur exécution pas à pas.

La commande TRACE affiche le Débogueur de 4D dans le process courant. La fenêtre du débogueur apparaît dès que la commande est appelée, avant l'exécution de la ligne de code suivante, et reste affichée pour l'exécution de chaque ligne de code. Vous pouvez également appeler manuellement le débogueur en utilisant la combinaison **Alt+Maj+clic droit** sous Windows ou **Control+Option+Commande+clic** sous Mac OS pendant l'exécution du code.

Dans les bases compilées, cette commande est ignorée.

4D Server : Si vous appelez TRACE depuis une méthode projet exécutée en tant que Procédure stockée, la fenêtre du débogueur apparaîtra sur le poste serveur.

Conseil : N'appelez pas TRACE lorsque vous utilisez un formulaire pour lequel les événements Sur activation et Sur désactivation ont été sélectionnés. En effet, chaque fois que la fenêtre du débogueur apparaîtra, les événements formulaire seront activés et cela créera une boucle sans fin entre les événements et le débogueur. De même, si vous appelez la commande TRACE depuis une méthode formulaire ou objet exécutée pendant la mise à jour du formulaire à l'écran, vous devrez également faire face à un problème de répétition sans fin de la séquence mises à jour du formulaire/apparitions de la fenêtre du débogueur.
Si vous vous retrouvez dans une telle situation, pour en sortir, utilisez la combinaison **Maj+clic** sur le bouton **Reprendre exécution** du débogueur. Tous les appels ultérieurs à TRACE dans le process seront ignorés.

Exemple

Dans le code suivant, la variable process CREER_LANG doit être égale à “US” ou “FR”. Si ce n'est pas le cas, la méthode projet DEBUG est appelée :

```
    ` ...
Au cas ou
    : (CREER_LANG="US")
        vsBHCmdNom:=[Commandes]CM US Nom
    : (CREER_LANG="FR")
        vsBHCmdNom:=[Commandes]CM FR Nom
Sinon
    DEBUG ("Valeur de CREER_LANG incorrecte")
Fin de cas
```

La méthode projet DEBUG est listée ci-dessous :

```
    ` Méthode projet DEBUG
    ` DEBUG (Texte)
    ` DEBUG (Informations supplémentaires de débogage)

    C_TEXTE ($1)

    Si (<>vbDebugOn) ` Variable interprocess définie dans la méthode base Sur ouverture
        Si (Mode compile)
            Si (Nombre de parametres>=1)
                ALERTE ($1+Caractere(13)+"Appelez le concepteur au 05 05 05 05")
            Fin de si
        Sinon
            TRACE
        Fin de si
    Fin de si
```

Référence

PAS DE TRACE.

Type (champVar) → Numérique

Paramètre	Type		Description
champVar	Champ | Variable	→	Champ ou variable à tester
Résultat	Numérique	←	Numéro du type de données

Description

Type retourne une valeur numérique qui indique le type du champ ou de la variable que vous avez passé(e) dans le paramètre champVar.

4D fournit les constantes prédéfinies suivantes :

Constante	Type	Valeur
Est un champ alpha	Entier long	0
Est une variable chaîne	Entier long	24
Est un texte	Entier long	2
Est un numérique	Entier long	1
Est un float	Entier long	35
Est un entier	Entier long	8
Est un entier long	Entier long	9
Est un entier 64 bits	Entier long	25
Est une date	Entier long	4
Est une heure	Entier long	11
Est un booléen	Entier long	6
Est une image	Entier long	3
Est une sous-table	Entier long	7
Est un BLOB	Entier long	30
Est une variable indéfinie	Entier long	5
Est un pointeur	Entier long	23
Est un tableau chaîne	Entier long	21
Est un tableau texte	Entier long	18
Est un tableau numérique	Entier long	14
Est un tableau entier	Entier long	15

Est un tableau entierlong	Entier long	16
Est un tableau date	Entier long	17
Est un tableau booléen	Entier long	22
Est un tableau image	Entier long	19
Est un tableau pointeur	Entier long	20
Est un tableau 2D	Entier long	13

Notes :
• Type retourne 9 (Est un entier long) lorsqu'elle est appliquée à une variable de type Graphe.
• A compter de la version 11 de 4D, Type retourne le véritable type d'un tableau lorsqu'elle est appliquée à une "ligne" d'un tableau 2D, et non plus Est un tableau 2D (cf. exemple 4).

Vous pouvez appliquer la fonction Type aux champs, variables interprocess, variables process, variables locales et à des pointeurs dépointés qui référencent ces types d'objets. Vous pouvez appliquer Type aux paramètres ($1,$2..., ${...}) d'une méthode projet ou au résultat d'une fonction ($0).

Exemples

(1) Référez-vous à l'exemple de la commande AJOUTER DONNEES AU CONTENEUR.

(2) Référez-vous à l'exemple de la commande PROPRIETES GLISSER DEPOSER.

(3) La méthode projet suivante efface une partie ou la totalité des champs de l'enregistrement courant de la table vers laquelle pointe le pointeur passé en paramètre, et ce, sans supprimer l'enregistrement ou changer d'enregistrement courant :

```
` Méthode projet VIDER ENREGISTREMENT
` VIDER ENREGISTREMENT ( Pointeur {; Entier long } )
` VIDER ENREGISTREMENT ( -> [Table] { ; Type des valeurs } )

C_POINTEUR ($1)
C_ENTIER LONG ($2;$vlTypeVal)

Si (Nombre de parametres>=2)
   $vlTypeVal:=$2
Sinon
   $vlTypeVal:=0xFFFFFFFF
Fin de si
Boucle ($vlChamp;1;Nombre de champs($1))
   $vpChamp:=Champ(Table($1);$vlChamp)
      $vlTypeChamp:=Type($vpChamp->)
```

```
        Si ( $vlTypeVal ?? $vlTypeChamp )
            Au cas ou
                : (($vlTypeChamp=Est un champ alpha)|($vlTypeChamp=Est un texte))
                    $vpChamp->:=""
                : (($vlTypeChamp=Est un numérique)|($vlTypeChamp=Est un entier)|
                                                        ($vlTypeChamp=Est un entier long))
                    $vpChamp->:=0
                : ($vlTypeChamp=Est une date)
                    $vpChamp->:=!00/00/00!
                : ($vlTypeChamp=Est une heure)
                    $vpChamp->:=?00:00:00?
                : ($vlTypeChamp=Est un booléen)
                    $vpChamp->:=Faux
                : ($vlTypeChamp=Est une image)
                    C_IMAGE($vgImageVide)
                    $vpChamp->:=$vgImageVide
                : ($vlTypeChamp=Est une sous table)
                    Repeter
                        TOUS LES SOUS ENREGISTREMENTS($vpChamp->)
                        SUPPRIMER SOUS ENREGISTREMENT($vpChamp->)
                    Jusque(Sous enregistrements trouves($vpChamp->)=0)
                : ($vlTypeChamp=Est un BLOB)
                    FIXER TAILLE BLOB($vpChamp->;0)
            Fin de cas
        Fin de si
    Fin de boucle
```

Une fois cette méthode projet implémentée dans votre base, vous pouvez écrire :

```
    ` Effacer la totalité du contenu de l'enregistrement courant de la table [Choses à faire]
VIDER ENREGISTREMENT (->[Choses à faire])

    ` Effacer les champs de type Texte, BLOB et Image de l'enregistrement courant de la
    ` table [Choses à faire]
VIDER ENREGISTREMENT (->[Choses à faire]; 0 ?+ Est un texte ?+ Est un BLOB ?+
                                                                    Est une image )

    ` Effacer la totalité de l'enregistrement courant de la table [Choses à faire] sauf
    ` les champs Alpha
VIDER ENREGISTREMENT (->[Choses à faire]; -1 ?- Est un champ alpha )
```

(4) Dans certains cas, par exemple pour écrire du code générique, il peut être nécessaire de savoir si un tableau est tableau standard indépendant ou une “ligne” d’un tableau 2D. Dans ce cas, il suffit d’utiliser le code suivant :

```
ptrmonTab:=->monTab{6}   ` monTab{6} est-il une ligne d’un tableau 2D ?
RESOUDRE POINTEUR(ptrmonTab;nomVar;numTable;numChamp)
Si(nomVar#"")
    $ptr:=Pointeur vers(nomVar)
    $letype:=Type($ptr->)
        ` Si monTab{6} est une ligne de tableau 2D, $letype vaut 13
Fin de si
```

Référence

Est une variable, Indefinie.

35

Liens

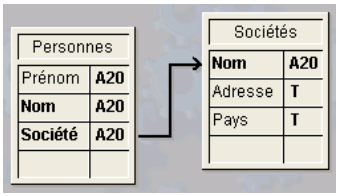
Les commandes de ce thème, en particulier CHARGER SUR LIEN et LIEN RETOUR, établissent et gèrent les relations entre les tables, à la fois pour les liens automatiques et les liens manuels. Consultez le manuel *Mode Développement* de 4D pour la création des liens entre les tables avant d'utiliser ces commandes.

Exploiter par programmation les liens automatiques entre les tables

Deux tables peuvent être reliées par un lien automatique. En général, quand un lien automatique est créé, les enregistrements liés sont chargés et sélectionnés dans la table liée. Un grand nombre d'opérations exploitent cette relation. En particulier, citons les opérations suivantes :
• Saisie de données,
• Liste des enregistrements à l'écran dans un formulaire de sortie,
• Etats,
• Opérations sur une sélection d'enregistrements comme les recherches, les tris et les formules.

Pour améliorer les performances, quand 4D active les liens automatiques, seul un enregistrement devient l'enregistrement courant pour la table. Pour chacune des opérations énumérées ci-dessus, l'enregistrement lié est chargé selon les principes suivants :
• Si un lien sélectionne un seul enregistrement de la table liée, cet enregistrement est chargé du disque.
• Si un lien sélectionne plusieurs enregistrements de la table liée, une nouvelle sélection d'enregistrements est créée pour cette table, et le premier enregistrement de cette sélection est chargé du disque.

Par exemple, dans la structure affichée ci-dessous, si un enregistrement pour la table [Personnes] est chargé et affiché pour une saisie de donnés, l'enregistrement lié dans la [Sociétés] est sélectionné et chargé. De la même façon, si un enregistrement pour la table [Sociétés] est chargé et affiché pour de la saisie de données, les enregistrements liés de la table [Personnes] sont sélectionnés.



Dans le type de schéma présenté ci-dessus, la table [Personnes] est appelée **Table N** et la table [Sociétés] est appelée **Table 1**. Pour vous souvenir de la différence, pensez à “plusieurs personnes travaillent dans une société” ou “une société emploie N personnes”.

De même, le champ Société de la table [Personnes] est appelé **Champ N**, et le champ Nom de la table [Sociétés] est appelé **Champ 1**.

Il n'est pas toujours possible que le champ lié soit unique. Par exemple, le champ [Sociétés]Nom peut contenir des noms de sociétés identiques. Ce cas de non-unicité est facilement contournable : il suffit de créer un lien vers un autre champ de la table liée qui, lui, sera toujours unique. Ce champ pourrait être par exemple un numéro d'identification de la société.

Les commandes listées ci-dessous utilisent les liens automatiques pour charger les enregistrements liés lors de leur exécution. Toutes ces commandes activent les liens automatiques "de N vers 1" existants. En revanche, seules les commandes signalées explicitement par un Oui activent les liens automatiques "de 1 vers N".

Commande	**Lien 1 vers N**
AJOUTER ENREGISTREMENT	Oui
AJOUTER SOUS ENREGISTREMENT	Non
APPLIQUER A SELECTION	Non
VISUALISER SELECTION	Non
ECRITURE DIF	Non
ECRITURE SYLK	Non
EXPORTER TEXTE	Non
EXPORTER DONNEES	Non
MODIFIER ENREGISTREMENT	Oui
MODIFIER SOUS ENREGISTREMENT	Non
MODIFIER SELECTION	Oui (en saisie de données)
TRIER	Non
TRIER PAR FORMULE	Non
CHERCHER PAR FORMULE	Oui
CHERCHER DANS SELECTION	Oui
CHERCHER	Oui
IMPRIMER ETIQUETTES	Non
IMPRIMER SELECTION	Oui
QR ETAT	Non
SELECTION VERS TABLEAU	Non
SELECTION LIMITEE VERS TABLEAU	Non

Activer par programmation les liens entre les tables

Que les liens soient automatiques ne signifie pas que les enregistrements liés ou les enregistrements pour une table seront sélectionnés simplement parce qu'une commande charge un enregistrement. Après avoir exécuté une commande chargeant un enregistrement, dans certains cas vous devez explicitement appeler le ou les enregistrement(s) lié(s) avec CHARGER SUR LIEN ou LIEN RETOUR si vous avez besoin d'accéder aux données liées.

Certaines des commandes listées ci-dessus (par exemple les commandes de recherche) chargent l'enregistrement courant une fois leur tâche terminée. Dans ce cas, l'enregistrement chargé n'appelle pas automatiquement le ou les enregistrement(s) lié(s). Là aussi, vous devez explicitement sélectionner le ou les enregistrement(s) lié(s) avec CHARGER SUR LIEN ou LIEN RETOUR si vous avez besoin d'accéder aux données liées.

Référence

ANCIEN LIEN RETOUR, CHARGER ANCIEN, CHARGER SUR LIEN, CREER SUR LIEN, FIXER LIEN CHAMP, FIXER LIENS AUTOMATIQUES, JOINTURE, LIEN RETOUR, LIRE LIEN CHAMP, LIRE LIENS AUTOMATIQUES, SELECTION RETOUR, STOCKER SUR LIEN.

ANCIEN LIEN RETOUR (champ)

Paramètre	Type		Description
champ	Champ	→	Champ recevant un lien

Description

ANCIEN LIEN RETOUR fonctionne comme la commande LIEN RETOUR, à la différence près que ANCIEN LIEN RETOUR utilise l'ancienne valeur du champ pour établir le lien.

Note : ANCIEN LIEN RETOUR utilise l'ancienne valeur du champ N, telle qu'elle est retournée par la fonction Ancien. Reportez-vous à la description de cette fonction pour plus d'informations.

ANCIEN LIEN RETOUR modifie la sélection de la table liée. La commande sélectionne le premier enregistrement de la sélection courante et en fait l'enregistrement courant.

Référence

CHARGER ANCIEN, LIEN RETOUR.

CHARGER ANCIEN (champ)

Paramètre	Type		Description
champ	Champ	→	Champ N

Description

CHARGER ANCIEN fonctionne de la même manière que CHARGER SUR LIEN, à la différence près que CHARGER ANCIEN utilise la valeur précédente de champ pour établir la relation.

Note : CHARGER ANCIEN utilise l'ancienne valeur du champ N telle qu'elle est retournée par la fonction Ancien. Reportez-vous à la description de cette fonction pour plus d'informations.

CHARGER ANCIEN charge l'enregistrement précédemment lié à l'enregistrement courant. Les champs de cet enregistrement sont alors saisissables. Si vous voulez modifier cet ancien enregistrement lié et le sauvegarder, vous devez appeler la commande STOCKER SUR LIEN. Notez que pour un enregistrement venant d'être créé, il n'y pas d'ancien enregistrement lié.

Référence

Ancien, ANCIEN LIEN RETOUR, CHARGER SUR LIEN, STOCKER SUR LIEN.

Variables et ensembles système

Si la commande a été correctement exécutée et si les enregistrements liés ont bien été chargés, la variable système *OK* prend la valeur 1. Si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue de choix d'enregistrement (qui apparaît si l'enregistrement lié avait été modifié), la variable *OK* prend la valeur 0.

version 11 (Modifiée)

CHARGER SUR LIEN (tableN | champN{; discriminant})

Paramètre	Type		Description
tableN | champN	Table | Champ	→	Table pour laquelle définir tous les liens automatiques ou Champ avec lien manuel partant vers la table 1
discriminant	Champ	→	Champ discriminant de la table 1

Description

CHARGER SUR LIEN accepte deux syntaxes.

La première syntaxe de la commande, CHARGER SUR LIEN(tableN), active tous les liens aller automatiques (de N vers 1) pour la table tableN dans le process courant. Cela signifie que pour chaque champ de la tableN d'où part un lien aller automatique, la commande sélectionnera l'enregistrement lié dans chaque table liée. Cela peut donc modifier l'enregistrement courant dans la (les) table(s) liée(s) du process courant.

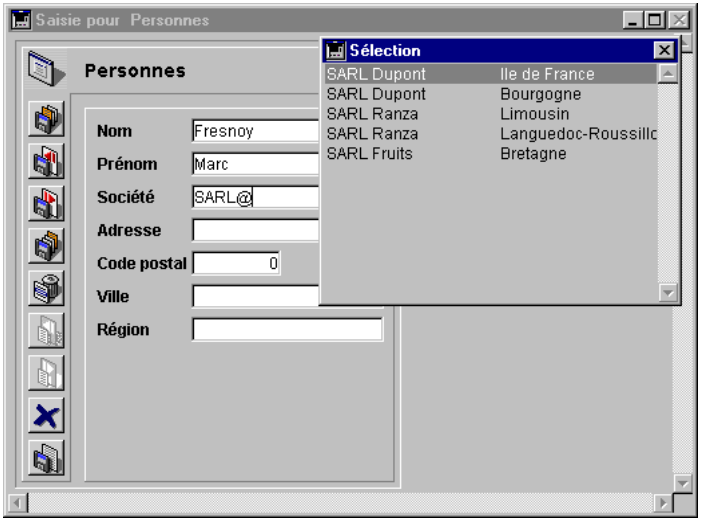
La seconde syntaxe, CHARGER SUR LIEN(champN{;discriminant}), recherche l'enregistrement lié au champ champN. Il n'est pas nécessaire que le lien soit automatique. S'il existe, CHARGER SUR LIEN charge en mémoire l'enregistrement lié, et en fait l'enregistrement et la sélection courants de la table à laquelle il appartient.

Le paramètre optionnel discriminant ne peut être spécifié que si champN est de type Alpha. Le champ discriminant doit être un champ de la table liée. Il peut être de type Alpha, numérique, Date, Heure ou Booléen. Autrement dit, il ne peut être du type Texte, Image, BLOB ou Sous-table.

Si champN est spécifié et si plus d'un enregistrement est trouvé dans la table liée, CHARGER SUR LIEN affiche une liste des enregistrements qui correspondent à la valeur de champN, permettant à l'utilisateur de sélectionner un enregistrement. Dans cette liste, la colonne de gauche affiche les valeurs des champs liés, la colonne de droite affiche les valeurs de discriminant.

Généralement, plusieurs enregistrements sont trouvés lorsque champN se termine par le caractère Joker (@). S'il n'y en a qu'un seul, la liste de sélection n'apparaît pas.

Dans l'écran ci-dessous, un enregistrement est en train d'être saisi et une liste de sélection s'affiche au premier plan.



La commande suivante a fait apparaître la liste de sélection :

CHARGER SUR LIEN ([Personnes]Société; [Sociétés]Région)

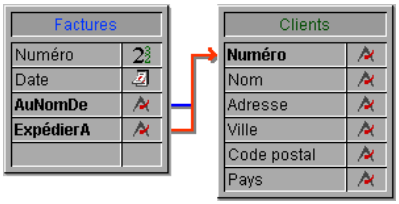
L'utilisateur a saisi SARL@ pour visualiser la liste de toutes les sociétés dont le nom commence par SARL, ainsi que leur région.

Spécifier un champ dans discriminant est la même opération que celle qui consiste à définir un champ discriminant dans la boîte de dialogue de définition des propriétés d'un lien en mode Développement. Pour plus d'informations sur la définition d'un champ discriminant, reportez-vous au manuel *Mode Développement* de 4D.

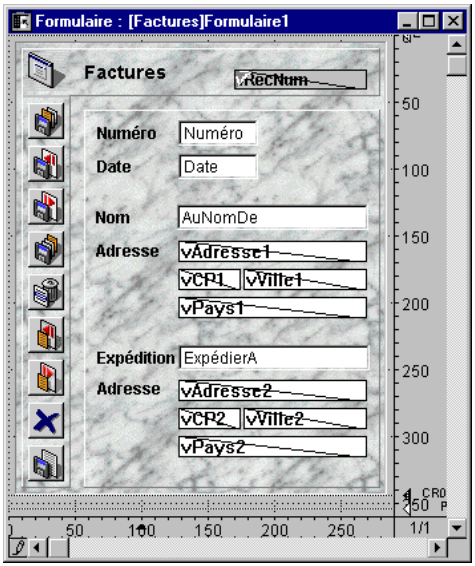
CHARGER SUR LIEN fonctionne avec les liens vers des sous-tables, mais il doit y avoir un lien vers la table parente et vers le champ lié de la sous-table pour que la relation fonctionne correctement. Lorsque vous utilisez une relation vers un sous-enregistrement, vous devez dans un premier temps appeler CHARGER SUR LIEN pour charger en mémoire l'enregistrement lié, puis appeler une seconde fois CHARGER SUR LIEN pour la sous-table.

Exemples

Dans l'exemple suivant, la table [Factures] est reliée à la table [Clients] par deux liens manuels. Un lien part du champ [Factures]AuNomDe et va vers le champ [Clients]Numéro, l'autre lien va de [Factures]ExpédierA à [Clients]Numéro.



Voici le formulaire de la table [Factures] affichant les informations "AuNomDe" et "ExpédierA".



Comme les deux liens pointent vers la même table, [Clients], l'information qu'ils récupèrent doit être affichée dans des variables. Si le formulaire contenait les champs de [Clients], seules les valeurs issues du second lien seraient affichées.

Les deux méthodes suivantes sont les méthodes objet des champs [Factures]ExpédierA et [Factures]AuNomDe. Voici la méthode objet du champ [Factures]AuNomDe :

```
CHARGER SUR LIEN ([Factures]AuNomDe; [Clients]Adresse)
vAdress1 := [Clients]Adresse
vVille1 := [Clients]Ville
vPays1 := [Clients]Pays
vCode1 := [Clients]CP
```

Voici la méthode objet du champ [Factures]ExpédierA :

```
CHARGER SUR LIEN ([Factures]ExpédierA; [Clients]Adresse)
vAdress2 := [Clients]Adresse
vVille2 := [Clients]Ville
vPays2 := [Clients]Pays
vCode2 := [Clients]CP
```

Référence

CHARGER ANCIEN, LIEN RETOUR.

Variables et ensembles système

Si la commande a été correctement exécutée et si les enregistrements liés ont bien été chargés, la variable système *OK* prend la valeur 1. Si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue de choix d'enregistrement (qui apparaît si l'enregistrement lié avait été modifié), la variable *OK* prend la valeur 0.

CREER SUR LIEN (champ)

Paramètre	Type		Description
champ	Champ	→	Champ N (champ d'où part le lien)

Description

CREER SUR LIEN a deux effets. S'il n'existe pas d'enregistrement lié à champ (c'est-à-dire si la valeur courante de champ n'est présente dans le champ correspondant d'aucun enregistrement de la table liée), CREER SUR LIEN crée un nouvel enregistrement lié. Si vous souhaitez conserver dans cet enregistrement la valeur de champ ayant provoqué sa création, assignez-la au champ correspondant. Utilisez ensuite la commande STOCKER SUR LIEN pour sauvegarder le nouvel enregistrement.

Si un enregistrement lié existe déjà, la commande CREER SUR LIEN a alors exactement le même effet que CHARGER SUR LIEN : elle charge en mémoire l'enregistrement lié.

Référence

STOCKER SUR LIEN.

FIXER LIEN CHAMP

FIXER LIEN CHAMP (tableN | champN; aller; retour)

Paramètre	Type		Description
tableN | champN	Table | Champ	→	Table de départ des liens ou Champ de départ du lien
aller	Entier long	→	Statut du lien aller partant du champ ou des liens aller partant de la table
retour	Entier long	→	Statut du lien retour partant du champ ou des liens retour partant de la table

Description

La commande FIXER LIEN CHAMP permet de définir séparément le statut automatique/manuel de chaque lien de la base pour le process courant, quel que soit son statut initial défini en mode Développement dans la fenêtre de paramétrage des liens.

Passez dans le premier paramètre un nom de table ou de champ :
• si vous passez un nom de champ (champN), la commande s'appliquera uniquement au lien partant du champ N désigné.
• si vous passez un nom de table (tableN), la commande s'appliquera à tous les liens partant de la table N désignée.
• si aucun lien ne part du champ champN ou de la table tableN, l'erreur de syntaxe n°16 ("Ce champ ne possède pas de lien") est générée et la variable système *OK* prend la valeur 0.

Passez dans les paramètres aller et retour des valeurs indiquant la modification du statut automatique/manuel à appliquer respectivement au(x) lien(s) de type N vers 1 — c'est-à-dire au(x) lien(s) aller — et au(x) lien(s) de type 1 vers N — c'est-à-dire au(x) lien(s) retour — désigné(s). Vous pouvez utiliser les constantes du thème "Liens" :
• Ne pas changer (0) = ne pas modifier le statut courant du ou des lien(s).
• Configuration Structure (1) = utiliser le paramétrage défini pour le(s) lien(s) dans la fenêtre de Structure de l'application.
• Manuel (2) = rendre manuel(s) le(s) lien(s) pour le process courant.
• Automatique (3) = rendre automatique(s) le(s) lien(s) pour le process courant.

Note : Les modifications effectuées à l'aide de cette commande s'appliquent au process courant uniquement. Le paramétrage des liens défini à l'aide des options de la fenêtre Inspecteur n'est pas modifié.

Exemple

Cette commande simplifie la gestion des liens avec l'éditeur d'états rapides. Dans les versions précédentes de 4D, pour utiliser les liens automatiques autres que ceux définis en mode Développement, il était nécessaire de passer tous les liens en automatique. Désormais, le code suivant permet de n'utiliser que les liens définis :

```
FIXER LIENS AUTOMATIQUES(Faux;Faux) `Initialisation des liens
    `Seuls les liens suivants seront utilisés
FIXER LIEN CHAMP([Facture]ID_Client;Automatique;Automatique)
FIXER LIEN CHAMP([Ligne_Facture]ID_Facture;Automatique;Automatique)
QR ETAT([Facture];Caractere(1);Vrai;Vrai;Vrai)
```

Référence

FIXER LIENS AUTOMATIQUES, LIRE LIEN CHAMP, LIRE LIENS AUTOMATIQUES, LIRE PROPRIETES LIEN.

FIXER LIENS AUTOMATIQUES

version 2004 (Modifiée)

FIXER LIENS AUTOMATIQUES (aller{; retour})

Paramètre	Type		Description
aller	Booléen	→	Statut de tous les liens de N vers 1
retour	Booléen	→	Statut de tous les liens de 1 vers N

Description

La commande FIXER LIENS AUTOMATIQUES transforme tous les liens manuels en liens automatiques pour toute la base dans le process courant. Cette modification est temporaire et peut à tout moment être remise en cause par un nouvel appel à FIXER LIENS AUTOMATIQUES.

• Si aller est Vrai, tous les liens N vers 1 deviennent automatiques. Si aller est Faux, tous les liens N vers 1 deviennent manuels.
• Si retour est Vrai, tous les liens 1 vers N deviennent automatiques. Si retour est Faux, tous les liens 1 vers N deviennent manuels.

Les liens définis comme automatiques en mode Développement ne sont pas affectés par cette commande. Si tous les liens sont définis comme manuels en mode Développement, cette commande vous permet de les rendre automatiques avant d'exécuter des opérations nécessitant qu'ils soient automatiques (par exemple, des recherches et tri relationnels). Après l'exécution de l'opération, le lien peut redevenir manuel.

Exemple

L'exemple suivant rend tous les liens N vers 1 automatiques et rétablit en manuel tous les liens 1 vers N qui étaient précédemment modifiés :

```
FIXER LIENS AUTOMATIQUES (Vrai; Faux)
```

Référence

FIXER LIEN CHAMP, LIRE LIENS AUTOMATIQUES, LIRE PROPRIETES LIEN, Présentation des liens, SELECTION LIMITEE VERS TABLEAU, SELECTION VERS TABLEAU.

JOINTURE (tableN; table1)

Paramètre	Type		Description
tableN	Table	→	Nom de la table N (d'où part le lien)
table1	Table	→	Nom de la table 1 (où arrive le lien)

Description

La commande JOINTURE crée une nouvelle sélection d'enregistrements dans table1 à partir de la sélection d'enregistrements de la tableN qui lui est liée et charge le premier enregistrement de la nouvelle sélection en tant qu'enregistrement courant.

Cette commande ne peut être utilisée que s'il existe un lien de N vers 1. JOINTURE peut opérer au travers de plusieurs niveaux de liens. Il peut y avoir plusieurs tables liées entre la table N et la table 1. Les liens peuvent être manuels ou automatiques.

Exemples

Nous souhaitons trouver tous les clients dont les factures arrivent à échéance aujourd'hui.

(1) L'exemple suivant propose une méthode pour créer une sélection dans la table [Clients] à partir d'une sélection d'enregistrements de la table [Factures] :

```
ENSEMBLE VIDE([Clients];"Paiement Du")
CHERCHER([Factures]; [Factures]PaiementDu=Date du jour)
Tant que (Non(Fin de selection([Factures])))
    CHARGER SUR LIEN([Factures]ClientID)
    ADJOINDRE ELEMENT([Clients];"Paiement Du")
ENREGISTREMENT SUIVANT([Factures])
Fin tant que
```

(2) L'exemple suivant parvient au même résultat que le précédent :

```
CHERCHER([Factures];[Factures]PaiementDu = Date du jour)
JOINTURE([Factures];[Clients])
```

Référence

CHARGER SUR LIEN, CHERCHER, Présentation des ensembles, SELECTION RETOUR.

LIEN RETOUR (table1 | champ1)

Paramètre	Type		Description
table1 | champ1	Table | champ	→	Table pour laquelle établir tous les liens de 1 vers N ou champ 1

Description

LIEN RETOUR a deux syntaxes.

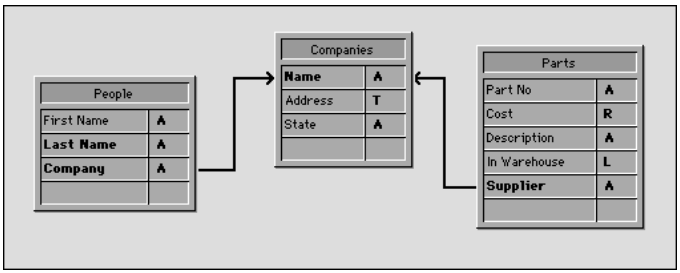
La première syntaxe, LIEN RETOUR (table1), établit tous les liens 1 vers N pour table1. Elle modifie la sélection courante pour chaque table qui a un lien 1 vers N vers table1. Les sélections courantes dans les tables N dépendent de la valeur courante de chaque champ lié dans la table 1. Chaque fois que cette commande est exécutée, les sélections courantes des tables N sont modifiées et le premier enregistrement de la sélection est chargé en tant qu'enregistrement courant.

La seconde syntaxe, LIEN RETOUR (champ1), établit le lien 1 vers N pour champ1. Elle modifie la sélection courante et l'enregistrement courant pour chaque table qui a un lien avec champ1. En conséquence, les enregistrements liés deviennent la sélection courante de la table N.

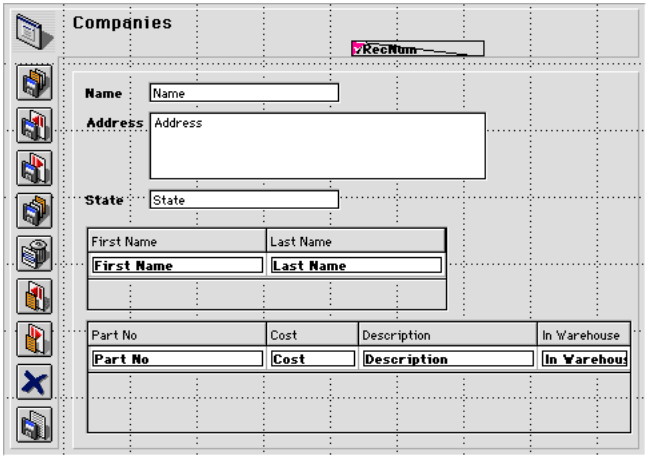
Notes :
• Si la sélection courante de la table 1 est vide au moment de l'exécution de LIEN RETOUR, la commande ne fait rien.
• Pour que la commande fonctionne, les champs clé d'appel (champs N) doivent être indexés.

Exemple

Dans l'exemple suivant, trois tables sont liées avec des liens automatiques. Les deux tables [People] et [Parts] ont un lien N vers 1 vers la table [Companies].



Voici le formulaire pour la table [Companies] qui affiche les enregistrements liés venant des tables [People] et [Parts].



Lorsque les formulaires pour People et Parts s'affichent, les enregistrements liés pour les tables [People] et [Parts] sont chargés et deviennent les sélections courantes de ces tables.

En revanche, les enregistrements liés ne sont pas chargés si un enregistrement de la table [Companies] est sélectionné par programmation. Dans ce cas, il faut utiliser la commande LIEN RETOUR.

Par exemple, la méthode suivante effectue une boucle sur chaque enregistrement de la table [Companies]. Pour chaque société, une alerte apparaît. Cette alerte affiche le nombre de personnes dans la société (le nombre d'enregistrements liés dans la table [People]) ainsi que le nombre de Parts que la société distribue (le nombre d'enregistrements dans la table [Parts] qui sont liés). Notez que nous avons besoin d'appeler la commande LIEN RETOUR bien que les liens soient automatiques :

```
TOUT SELECTIONNER ([Companies]) ` Sélectionner tous les enregistrements dans la table
TRIER ([Companies]; [Companies]Nom)
    ` Trier les enregistrements dans l'ordre alphabétique
Boucle ($i; 1; Enregistrements dans table ([Companies]))
        ` Boucler une fois par enregistrement
    LIEN RETOUR ([Companies]Nom) ` Sélectionner les enregistrements liés
    ALERTE ("Société : "+[Companies]Nom + Caractere (13) +"personnes dans la société : "
            + Chaine (Enregistrements trouves ([People])) + Caractere(13) + "Nombre de
                Produits qu'ils distribuent : "+ Chaine (Enregistrements trouves ([Parts])))
    ENREGISTREMENT SUIVANT ([Companies]) ` Aller à l'enregistrement suivant
Fin de boucle
```

Référence

ANCIEN LIEN RETOUR, CHARGER SUR LIEN.

LIRE LIEN CHAMP (champN; aller; retour{; *})

Paramètre	Type		Description
champN	Champ	→	Champ de départ du lien
aller	Entier long	←	Statut du lien aller
retour	Entier long	←	Statut du lien retour
*	*	→	• Si passé : aller et retour retournent le statut courant effectif du lien (valeurs 2 ou 3 uniquement) • Si omis (défaut) : aller et retour peuvent retourner la valeur 1 si le lien n'a pas été modifié par programmation

Description

La commande LIRE LIEN CHAMP permet de connaître le statut automatique/manuel du lien partant du champN pour le process courant. Tous les liens peuvent être consultés, y compris les liens déclarés automatiques dans la fenêtre de Structure.

• Passez dans champN le nom du champ de la table N d'où part le lien dont vous souhaitez connaître le statut. Si aucun lien ne part du champ champN, les paramètres aller et retour retournent 0, une erreur est générée et la variable système *OK* prend la valeur 0 (cf. ci-dessous).

• Après l'exécution de la commande, la variable aller contient une valeur indiquant si le lien aller spécifié est défini comme automatique :

0 = il n'y a pas de lien partant de champN. L'erreur de syntaxe n°16 ("Ce champ ne possède pas de lien") est générée et la variable système *OK* prend la valeur 0.

1 = le statut automatique/manuel du lien aller spécifié est celui défini par l'option **Lien aller auto** dans les propriétés du lien en mode Développement (il n'a pas été modifié par programmation).

2 = le lien N vers 1 est manuel pour le process.

3 = le lien N vers 1 est automatique pour le process.

• Après l'exécution de la commande, la variable retour contient une valeur indiquant si le lien retour spécifié est défini comme automatique :

0 = il n'y a pas de lien partant de champN. L'erreur de syntaxe n°16 ("Ce champ ne possède pas de lien") est générée et la variable système *OK* prend la valeur 0.

1 = le statut automatique/manuel du lien retour spécifié est celui défini par l'option **Lien retour auto** dans les propriétés du lien en mode Développement (il n'a pas été modifié par programmation).

2 = le lien 1 vers N est manuel pour le process.

3 = le lien 1 vers N est automatique pour le

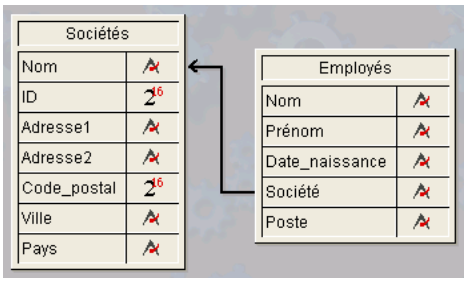
Vous pouvez comparer les valeurs reçues dans les paramètres aller et retour aux constantes du thème “Liens” :

Constante	Type	Valeur
Pas de lien	Entier long	0
Configuration Structure	Entier long	1
Manuel	Entier long	2
Automatique	Entier long	3

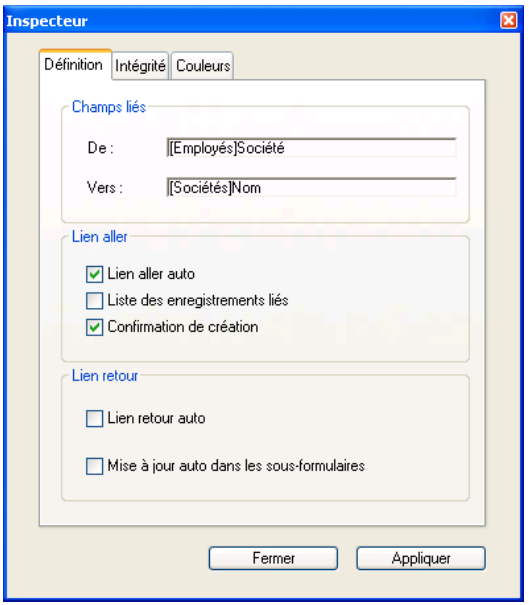
• Le paramètre facultatif * permet de “forcer” la lecture du statut courant du lien, même s’il n’a pas été modifié par programmation. Autrement dit, lorsque vous passez le paramètre *, seules les valeurs 2 ou 3 peuvent être retournées dans les paramètres aller et retour.

Exemple

Soit la structure suivante :



Les propriétés du lien reliant le champ [Employés]Société au champ [Sociétés]Nom sont les suivantes :



Le code ci-dessous illustre les différentes possibilités offertes par les commandes LIRE LIEN CHAMP, LIRE LIENS AUTOMATIQUES, FIXER LIEN CHAMP et FIXER LIENS AUTOMATIQUES ainsi que leurs effets :

```
LIRE LIENS AUTOMATIQUES(liens_Appel;liens_Retour)  `retourne Faux, Faux
LIRE LIEN CHAMP([Employés]Société;aller;retour)  `retourne 1,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*)  `retourne 3,2

FIXER LIEN CHAMP ([Employés]Société;2;0)  `passe le lien N vers 1 en manuel

LIRE LIEN CHAMP([Employés]Société;aller;retour) `retourne 2,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*) `retourne 2, 2

FIXER LIEN CHAMP ([Employés]Société;1;0)  `rétablit les paramètres définis en
                                          `structure pour le lien N vers 1

LIRE LIEN CHAMP([Employés]Société;aller;retour)  `retourne 1,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*)  `retourne 3,2

FIXER LIENS AUTOMATIQUES(Vrai;Vrai)
   `passe tous les liens de toutes les tables en automatique

LIRE LIENS AUTOMATIQUES(liens_Appel;liens_Retour) `retourne Vrai, Vrai
LIRE LIEN CHAMP([Employés]Société;aller;retour)  `retourne 1,1
LIRE LIEN CHAMP([Employés]Société;aller;retour;*)  `retourne 3,3
```

Référence

FIXER LIEN CHAMP, FIXER LIENS AUTOMATIQUES, LIRE LIENS AUTOMATIQUES, LIRE PROPRIETES LIEN.

LIRE LIENS AUTOMATIQUES

version 2004

LIRE LIENS AUTOMATIQUES (aller; retour)

Paramètre	Type		Description
aller	Booléen	←	Statut de tous les liens de N vers 1
retour	Booléen	←	Statut de tous les liens de 1 vers N

Description

La commande LIRE LIENS AUTOMATIQUES permet de savoir si le statut automatique/manuel de tous les liens manuels N vers 1 et 1 vers N de la base a été modifié dans le process courant.

• aller : ce paramètre retourne Vrai si un appel antérieur de la commande FIXER LIENS AUTOMATIQUES a rendu automatiques tous les liens manuels N vers 1 — par exemple FIXER LIENS AUTOMATIQUES(Vrai;Faux).
Ce paramètre retourne Faux si la commande FIXER LIENS AUTOMATIQUES n'a pas été appelée ou si sa précédente exécution n'a pas modifié les liens manuels N vers 1 — par exemple FIXER LIENS AUTOMATIQUES(Faux;Faux).

• retour : ce paramètre retourne Vrai si l'appel précédent de la commande FIXER LIENS AUTOMATIQUES a rendu automatiques tous les liens manuels 1 vers N — par exemple FIXER LIENS AUTOMATIQUES(Vrai;Vrai).
Ce paramètre retourne Faux si la commande FIXER LIENS AUTOMATIQUES n'a pas été appelée ou si sa précédente exécution n'a pas modifié les liens manuels 1 vers N — par exemple FIXER LIENS AUTOMATIQUES(Vrai;Faux).

Exemple

Reportez-vous à l'exemple de la commande LIRE LIEN CHAMP.

Référence

FIXER LIENS AUTOMATIQUES, LIRE LIEN CHAMP, LIRE PROPRIETES LIEN.

SELECTION RETOUR (leChamp)

Paramètre	Type		Description
leChamp	Champ	→	Champ de la table N (d'où part le lien)

Description

La commande SELECTION RETOUR crée une sélection d'enregistrements dans la table N basée sur la sélection courante de la table 1, et charge le premier enregistrement de la table N comme enregistrement courant.

Note : SELECTION RETOUR modifie l'enregistrement courant de la table 1.

Exemple

Prenons l'exemple d'une base de données comportant une table [Factures] dont le champ [Factures]IDClient est lié au champ [Clients]NoID de la table [Clients]. L'exemple suivant sélectionne toutes les factures adressées aux clients dont le crédit est supérieur ou égal à 5710 Euros :

```
    ` Sélectionner les clients
CHERCHER ([Clients];[Clients]Credit>=5710)
    `Trouver toutes les factures liées à chacun de ces clients
SELECTION RETOUR ([Factures]IDClient)
```

Référence

CHARGER SUR LIEN, CHERCHER, JOINTURE.

STOCKER SUR LIEN

STOCKER SUR LIEN (champ)

Paramètre	Type		Description
champ	Champ	→	Champ N

Description

STOCKER SUR LIEN sauvegarde l'enregistrement lié à champ. Vous pouvez exécuter une commande STOCKER SUR LIEN pour mettre à jour un enregistrement créé par CREER SUR LIEN, ou bien lorsque vous voulez sauvegarder des modifications apportées à un enregistrement chargé par CHARGER SUR LIEN.

STOCKER SUR LIEN ne s'applique pas aux sous-tables car la sauvegarde d'un enregistrement parent entraîne automatiquement la sauvegarde des sous-enregistrements.

STOCKER SUR LIEN ne sauvegardera pas un enregistrement verrouillé. Lorsque vous appelez cette commande, vous devez tout d'abord vous assurer que l'enregistrement n'est pas verrouillé. S'il est verrouillé, la commande est ignorée, l'enregistrement n'est pas sauvegardé et aucune erreur ne vous est retournée.

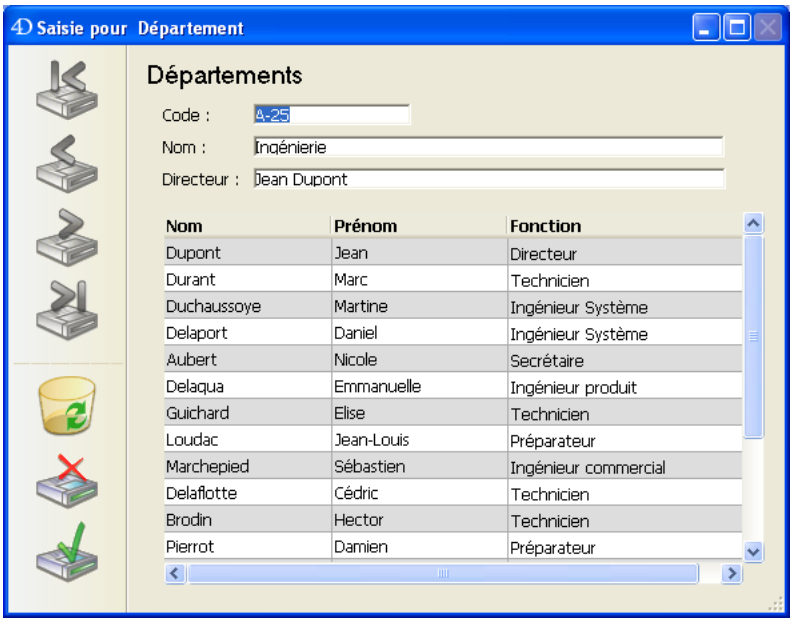
Référence

CHARGER SUR LIEN, CREER SUR LIEN, Enregistrement verrouille, Présentation des triggers.

36

List Box

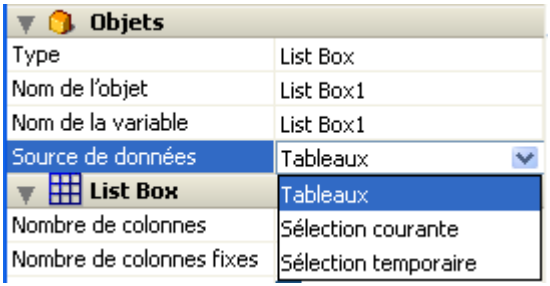
Les commandes de ce thème sont dédiées à la gestion programmée des objets de formulaire de type List box.
Les list box peuvent être comparées aux Zones de défilement groupées. De fait, une list box propose toutes les fonctions d'un ensemble de zones de défilement groupées, notamment la possibilité de représenter des données sous forme de colonnes et de lignes sélectionnables. Les list box proposent en outre de nombreuses fonctions supplémentaires telles que la possibilité de saisir des valeurs, trier les colonnes, définir des couleurs alternées, etc.



Un objet de type List box est entièrement paramétrable dans l'éditeur de formulaires de 4D et peut également être contrôlé par programmation. Pour plus d'informations sur la création et le paramétrage des objets de type List box dans l'éditeur de formulaires ainsi que leur utilisation, reportez-vous au manuel *Mode Développement* de la documentation de 4D.
La programmation des objets de type List box s'effectue sur le même modèle que les autres objets de formulaire en liste de 4D. Elle doit cependant tenir compte de principes spécifiques, décrits dans cette section.

Note : Les list box sont des objets destinés à l'interface écran. Il n'est pas possible de les imprimer.

Un objet List box peut contenir une ou plusieurs colonnes et peut être associé soit à des tableaux 4D, soit à une sélection d'enregistrements. Dans le cas des list box de type sélection, les colonnes sont associées à des champs ou des expressions.
Il n'est pas possible de combiner dans une même list box ces deux types de sources de données (tableaux et sélections). La définition de la source de données s'effectue au moment de la création de l'objet List box dans l'éditeur de formulaires, via la Liste des propriétés. Il n'est pas possible de la modifier ensuite par programmation.



List box de type tableau
Dans ce type de list box, chaque colonne est associée à un tableau 4D à une dimension ; tous les types de tableaux peuvent être utilisés, à l'exception des tableaux de pointeurs. Le format d'affichage de chaque colonne peut être défini dans l'éditeur de formulaires ou via la commande CHOIX FORMATAGE.

En mode programmé, les valeurs des colonnes (saisie et affichage) sont gérées à l'aide des commandes de haut niveau du thème List box (telles que INSERER LIGNE LISTBOX ou SUPPRIMER LIGNE LISTBOX) ainsi que des commandes de manipulation des tableaux.
Par exemple, pour initialiser le contenu d'une colonne de List box, vous pouvez utiliser l'instruction suivante :

TABLEAU TEXTE(NomColonne; taille)

Vous pouvez également utiliser une énumération :

ENUMERATION VERS TABLEAU("NomEnum"; NomColonne)

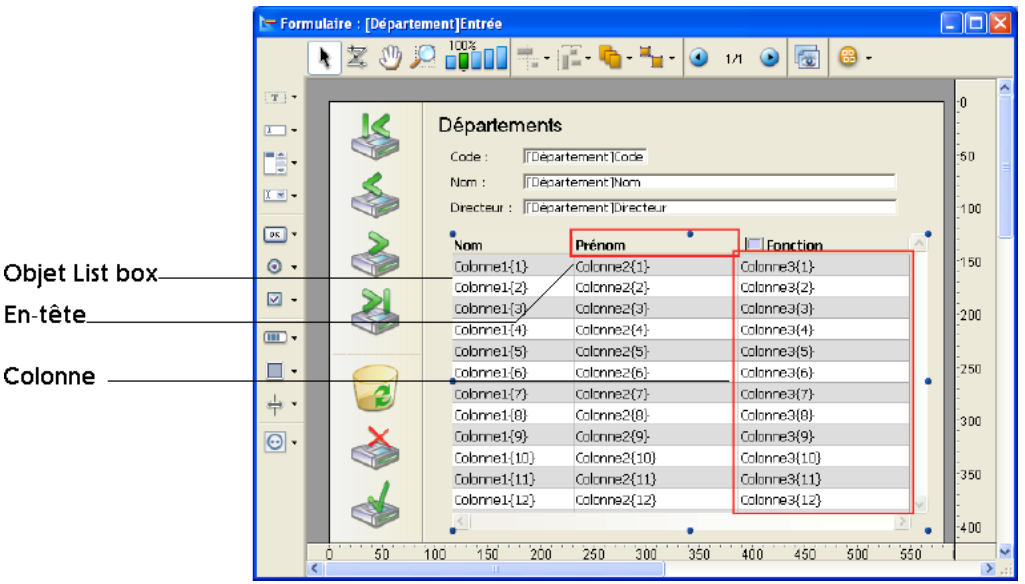
Attention : Lorsqu'un objet List box contient plusieurs colonnes de tailles différentes, seul le nombre d'éléments correspondant au plus petit tableau est affiché. Il est donc conseillé de veiller à ce que chaque tableau ait le même nombre d'éléments que les autres. A noter également que si une colonne de la list box est "vide" (c'est le cas lorsque le tableau associé n'a pas été déclaré ou dimensionné via le langage), la list box n'affiche aucun contenu.

List box de type sélection
Dans ce type de list box, chaque colonne peut être associée à un champ ou à une expression. Le contenu de chaque ligne est alors évalué en fonction d'une sélection d'enregistrements : la sélection courante d'une table ou une sélection temporaire.

Dans le cas de la sélection courante, toute modification effectuée côté base de données est automatiquement reportée dans la list box et inversement. La sélection courante est donc toujours identique aux deux emplacements. A noter que les commandes INSERER LIGNE LISTBOX et SUPPRIMER LIGNE LISTBOX ne peuvent pas être utilisées avec les list box de type sélection.
Vous pouvez associer une colonne de list box à une expression. L'expression pourra être basée sur un ou plusieurs champs (par exemple [Employés]Nom+" "+[Employés]Prénom) ou être simplement une formule (par exemple Chaine(Millisecondes)). L'expression peut également être une méthode projet, une variable ou un élément de tableau.
La commande FIXER TABLE SOURCE LISTBOX permet de modifier par programmation la table associée à la list box.

Objet, colonne et en-tête

Un objet List box est composé de trois types d'éléments distincts :
• l'**objet** dans son ensemble,
• les **colonnes**,
• les **en-têtes** des colonnes.
Dans l'éditeur de formulaires, ces éléments peuvent être sélectionnés séparément. Chacun d'eux dispose de son propre nom d'objet et nom de variable et peut donc être adressé séparément.



Par défaut, les colonnes sont nommées Colonne1 à n et les en-têtes Entête1 à n dans le formulaire, indépendamment des objets List box eux-mêmes. A noter que, par défaut, le même nom est utilisé pour les objets et leurs variables associées.

Chaque type d'élément dispose de caractéristiques propres et de caractéristiques partagées avec les autres éléments. Par exemple, la police de caractères peut être assignée globalement à l'objet List box ou séparément aux colonnes et aux en-têtes. A l'inverse, les propriétés de saisie ne sont définissables que pour les colonnes.
Ces principes s'appliquent aux commandes du thème "Propriétés des objets" pouvant être utilisées avec les list box : en fonction de sa nature, chaque commande sera utilisable avec la list box, les colonnes et/ou les en-têtes des colonnes. Pour désigner le type d'élément sur lequel vous souhaitez agir, il suffit de passer le nom ou la variable qui lui est associé(e).
Le tableau suivant précise la portée de chaque commande du thème "Propriétés des objets" utilisable avec les objets de type list box :

Commandes Propriétés des objets	Objet	Colonne	En-tête de colonne
DEPLACER OBJET	X		
LIRE RECT OBJET	X		
CHOIX FILTRE SAISIE		X	
CHOIX FORMATAGE		X	
CHOIX SAISISSABLE		X	
CHOIX ENUMERATION		X	
TITRE BOUTON			X
CHOIX COULEUR	X	X	X
FIXER COULEURS RVB	X	X	X
CHANGER JEU DE CARACTERES	X	X	X
CHANGER TAILLE	X	X	X
CHANGER STYLE	X	X	X
FIXER ALIGNEMENT	X	X	X
Lire alignement	X	X	X
CHOIX VISIBLE	X	X	X
CHOIX VISIBLE BARRES DEFILEMENT	X		
TAILLE OBJET OPTIMALE	X	X	X

Notes :
• Toutes les commandes du thème "List Box" s'appliquent à l'objet List box seulement, à l'exception de FIXER LARGEUR COLONNE LISTBOX (objet, colonne et en-tête) et Lire largeur colonne listbox (colonne ou en-tête uniquement).
• Avec les List box de type tableau, il est possible de définir séparément les propriétés de style, de couleur de police, de couleur de fond et de visibilité de chaque ligne. Cette gestion s'effectue via des tableaux associés à la list box dans la Liste des propriétés. Vous pouvez récupérer par programmation le nom de ces tableaux à l'aide de la commande LIRE TABLEAUX LISTBOX.

List box et Langage

Méthodes objet

Il est possible d'associer une méthode à l'objet List box dans son ensemble et/ou à chaque colonne de la list box. Les méthodes objet sont appelées dans l'ordre suivant :
1. Méthode objet de la colonne
2. Méthode objet de la list box

La méthode objet de la colonne reçoit les événements se produisant dans son en-tête.

CHOIX VISIBLE et en-têtes
Lorsque la commande CHOIX VISIBLE est utilisée avec un en-tête de list box, elle agit sur tous les en-têtes de l'objet List box, quel que soit l'en-tête spécifié par la commande. Par exemple, l'instruction CHOIX VISIBLE(*;"entête3";Faux) masquera tous les en-têtes de l'objet List box auquel appartient l'en-tête 3 (et non uniquement cet en-tête).

Objet focus et Self
Les fonctions Objet focus (thème "Interface utilisateur") et Self (thème "Langage") peuvent être utilisées dans la méthode objet d'une list box ou d'une colonne de list box. Elles retournent un pointeur vers la list box, la colonne de list box (1) ou la variable d'en-tête en fonction du type d'événement formulaire. Le tableau suivant détaille ce fonctionnement :

Evénement	**Objet focus**	**Self**
Sur clic	list box	colonne
Sur double clic	list box	colonne
Sur avant frappe clavier	colonne	colonne
Sur après frappe clavier	colonne	colonne
Sur après modification	colonne	colonne
Sur gain focus	colonne ou list box (*)	colonne ou list box (*)
Sur perte focus	colonne ou list box (*)	colonne ou list box (*)
Sur déposer	list box source	list box (*)
Sur glisser	list box source	list box (*)
Sur début glisser	list box	list box (*)
Sur début survol	list box (**)	list box (**)
Sur survol	list box (**)	list box (**)
Sur fin survol	list box (**)	list box (**)
Sur données modifiées	colonne	colonne
Sur nouvelle sélection	list box (**)	list box (**)
Sur avant saisie	colonne	colonne
Sur déplacement colonne	list box	colonne
Sur déplacement ligne	list box	list box
Sur redimensionnement colonne	list box	colonne
Sur clic entête	list box	en-tête
Sur après tri	list box	en-tête

(*) Lorsque le focus est modifié à l'intérieur d'une list box, un pointeur vers la colonne est retourné. Lorsque le focus est modifié au niveau global du formulaire, un pointeur vers la list box est retourné. Dans le contexte d'une méthode objet de colonne, un pointeur vers la colonne est retourné.
(**) Non exécuté dans le contexte d'une méthode objet de colonne.

(1) Lorsqu'un pointeur vers la colonne est retourné, l'objet pointé dépend du type de la list box. Dans le cadre d'une list box de type tableau, Objet focus et Self retournent un pointeur vers le tableau. Le mécanisme des pointeurs de 4D permet alors de connaître le numéro de l'élément de tableau modifié.

Par exemple, en supposant que l'utilisateur a modifié la 5e ligne de la colonne col2 :

```
$Colonne:=Objet focus
    ` $Colonne contient un pointeur vers col2
$Ligne:= $Colonne->  `$Ligne vaut 5
```

Dans le cadre d'une list box de type sélection, Objet focus et Self retournent :
• pour une colonne associée à un champ, un pointeur vers le champ associé,
• pour une colonne associée à une variable, un pointeur vers la variable,
• pour une colonne associée à une expression, un pointeur Nil.

DEFILER LIGNES
Il est possible d'utiliser la commande DEFILER LIGNES (thème "Interface utilisateur") avec un objet de type list box. Cette commande permet de faire défiler les lignes de la list box afin d'afficher la première ligne sélectionnée ou une ligne spécifique.

EDITER ELEMENT
La commande EDITER ELEMENT (thème "Gestion de la saisie") permet de passer en mode édition une cellule d'un objet list box.

Numero de ligne affichee
La commande Numero de ligne affichee (thème "Sélections") fonctionne dans le contexte de l'événement Sur affichage corps pour un objet list box.

Evénements formulaire
Des événements formulaire spécifiques sont destinés à la gestion programmée des list box, concernant notamment le glisser-déposer et le tri. Pour plus d'informations, reportez-vous à la description de la commande Evenement formulaire.

Glisser déposer
La gestion du glisser-déposer de données dans les list box est prise en charge par les commandes Position deposer et PROPRIETES GLISSER DEPOSER. Ces commandes ont été spécialement adaptées pour les list box.
Attention de ne pas confondre le glisser-déposer avec le déplacement de lignes et de colonnes, pris en charge par les commandes NUMERO LIGNE LISTBOX DEPLACEE et NUMERO COLONNE LISTBOX DEPLACEE.

Gestion des tris

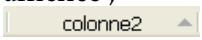
Par défaut, la list box gère automatiquement les tris standard des colonnes en cas de clic sur l'en-tête. Un tri standard est un tri alphanumérique des valeurs de la colonne, alternativement croissant / décroissant lors de clics multiples. Toutes les colonnes sont toujours automatiquement synchronisées.
Il est possible d'interdire le tri utilisateur standard en désélectionnant la propriété "Triable" pour la list box.

Le développeur peut mettre en place une gestion personnalisée des tris à l'aide de la commande TRIER COLONNES LISTBOX et/ou en combinant les événements formulaire Sur clic entête et Sur après tri (cf. commande Evénement formulaire) et les commandes 4D de gestion des tableaux.

Note : La propriété “Triable” concerne uniquement le tri utilisateur standard, la commande TRIER COLONNES LISTBOX ne tient pas compte de cette propriété.

En outre, la valeur de la variable associée à l’en-tête d’une colonne permet de gérer une information supplémentaire : le tri courant de la colonne (lecture) et l’affichage de la flèche de tri.
• si la variable vaut 0, la colonne n’est pas triée et la flèche de tri n’est pas affichée ;

• si la variable vaut 1, la colonne est triée par ordre croissant et la flèche de tri croissant est affichée ;



• si la variable vaut 2, la colonne est triée par ordre décroissant et la flèche de tri décroissant est affichée.

Il est possible de fixer la valeur de la variable (par exemple Header2:=2) afin de “forcer” l’affichage de la flèche de tri. Le tri de la colonne lui-même n’est dans ce cas pas modifié, il appartient au développeur de le gérer.

Gestion des sélections

La gestion des sélections s'effectue différemment pour les list box de type tableau et de type sélection.

• **List box de type sélection** : les sélections sont gérées par l'intermédiaire d'un ensemble appelé "Ensemble surlignage". Cet ensemble est défini dans les propriétés de la list box. Il est maintenu automatiquement par 4D : si l'utilisateur sélectionne une ou plusieurs ligne(s) dans la list box, l'ensemble est immédiatement mis à jour). A l'inverse, il est possible d'utiliser les commandes du thème "Ensembles" afin de modifier par programmation la sélection dans la list box.

• **List box de type tableau** : la commande SELECTIONNER LIGNE LISTBOX permet de sélectionner par programmation une ou plusieurs lignes de list box.
En outre, la variable associée à l’objet List box peut être utilisée pour lire, fixer ou stocker les sélections de lignes dans l’objet. Cette variable correspond à un tableau de booléens automatiquement créé et maintenu par 4D. La taille de ce tableau est déterminée par celle de la list box : il contient le même nombre d’éléments que le plus petit tableau associé aux colonnes.

Chaque élément de ce tableau contient Vrai si la ligne correspondante est sélectionnée et Faux sinon. 4D met à jour le contenu de ce tableau en fonction des actions utilisateur. A l'inverse, vous pouvez modifier la valeur des éléments de ce tableau afin de modifier la sélection dans la list box.
En revanche, vous ne pouvez ni insérer ni supprimer de ligne dans ce tableau ; il n'est pas possible non plus de le retyper.

Note : La commande Compter dans tableau est utile dans ce cas pour connaître le nombre de lignes sélectionnées.

Par exemple, cette méthode permet d'inverser la sélection de la première ligne de la list box (type tableau) :

```
TABLEAU BOOLEEN(tBListBox;10)
    ` tBListBox est le nom de la variable associée à la List box dans le formulaire
Si (tBListBox{1} = Vrai)
    tBListBox{1}:= Faux
Sinon
    tBListBox{1}:= Vrai
Fin de si
```

FIXER COULEUR GRILLE LISTBOX

version 2004

FIXER COULEUR GRILLE LISTBOX ({*; }objet; couleur; horizontal; vertical)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
couleur	Numérique	→	Valeur de couleur RVB
horizontal	Booléen	→	Utiliser la couleur pour les traits horizontaux
vertical	Booléen	→	Utiliser la couleur pour les traits verticaux

Description

La commande FIXER COULEUR GRILLE LISTBOX permet de modifier la couleur de la grille de l'objet list box désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Passez dans le paramètre couleur une valeur de couleur RVB. Pour plus d'informations sur les couleurs RVB, reportez-vous à la description de la commande FIXER COULEURS RVB.

Les paramètres horizontal et vertical vous permettent de spécifier les traits auxquels la couleur doit être appliquée :
• si vous passez Vrai dans horizontal, la couleur sera appliquée aux traits horizontaux de la grille. Si vous passez Faux, leur couleur ne sera pas modifiée.
• si vous passez Vrai dans vertical, la couleur sera appliquée aux traits verticaux de la grille. Si vous passez Faux, leur couleur ne sera pas modifiée.

Référence

FIXER COULEURS RVB, MONTRER GRILLE LISTBOX.

FIXER HAUTEUR LIGNES LISTBOX

FIXER HAUTEUR LIGNES LISTBOX ({*; }objet; hauteur)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
hauteur	Entier	→	Hauteur de ligne (en pixels)

Description

La commande FIXER HAUTEUR LIGNES LISTBOX permet de modifier par programmation la hauteur des lignes de l'objet list box désigné par les paramètres objet et *.

Si vous passez le paramètre facultatif *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Référence

Lire hauteur lignes listbox.

FIXER LARGEUR COLONNE LISTBOX

version 2004

FIXER LARGEUR COLONNE LISTBOX ({*; }objet; largeur)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
largeur	Entier	→	Largeur de colonne (en pixels)

Description

La commande FIXER LARGEUR COLONNE LISTBOX permet de modifier par programmation la largeur d'une ou de toutes les colonne(s) de l'objet (list box, colonne ou en-tête) désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Passez dans le paramètre largeur la nouvelle largeur (en pixels) de l'objet.
• Si objet désigne l'objet list box, toutes les colonnes de la list box sont redimensionnées.
• Si objet désigne une colonne ou un en-tête de colonne, seule la colonne désignée est redimensionnée.

Référence

Lire largeur colonne listbox.

FIXER TABLE SOURCE LISTBOX

version 11

FIXER TABLE SOURCE LISTBOX ({*; }objet; numTable | tempo)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
numTable | tempo	Entier long |Chaîne	→	Numéro de la table de laquelle utiliser la sélection courante ou Nom de la sélection temporaire à utiliser

Description

La commande FIXER TABLE SOURCE LISTBOX vous permet de modifier la source des données affichées dans la list box désignée par les paramètres * et objet.

Note : Cette commande ne peut être utilisée que lorsque la propriété "Source de données" de la list box est **Sélection courante** ou **Sélection temporaire** (pour plus d'informations sur ce point, reportez-vous à la section Gestion programmée des objets de type List box). Elle ne fait rien si vous l'utilisez avec une list box associée à des tableaux.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Si vous passez un numéro de table comme paramètre numTable, la list box sera remplie avec les données des enregistrements de la sélection courante de la table.
Si vous passez un nom de sélection temporaire comme paramètre tempo, la list box sera remplie avec les données des enregistrements appartenant à la sélection temporaire.

Si la list box contenait déjà des colonnes, leur contenu est mis à jour à l'issue de l'exécution de la commande.

Note : Pour des raisons d'optimisation, cette commande est traitée de manière asynchrone, c'est-à-dire que le changement de source de la listbox n'est effectif qu'à l'issue de l'exécution complète de la méthode dans laquelle la commande est appelée.

Référence

LIRE TABLE SOURCE LISTBOX.

INSERER COLONNE FORMULE LISTBOX

version 11

INSERER COLONNE FORMULE LISTBOX ({*; }objet; positionCol; nomCol; formule; typeDonnées; nomEntête; variableEntête)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionCol	Numérique	→	Emplacement de la colonne à insérer
nomCol	Chaîne	→	Nom d'objet de la colonne
formule	Chaîne	→	Formule 4D associée à la colonne
typeDonnées	Entier long	→	Type de résultat de la formule
nomEntête	Chaîne	→	Nom d'objet de l'en-tête de la colonne
variableEntête	Var Entier	→	Variable d'en-tête de la colonne

Description

La commande INSERER COLONNE FORMULE LISTBOX insère une colonne dans la list box désignée par les paramètres objet et *.
La commande INSERER COLONNE FORMULE LISTBOX est semblable à la commande INSERER COLONNE LISTBOX, à la différence près qu'elle permet la saisie d'une formule comme contenu de la colonne.
Ce type de contenu ne peut être utilisé que lorsque la propriété "Source de données" de la list box est **Sélection courante** ou **Sélection temporaire** (pour plus d'informations sur ce point, reportez-vous à la section Gestion programmée des objets de type List box).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La nouvelle colonne est insérée juste avant la colonne désignée par le paramètre positionCol. Si le paramètre positionCol est supérieur au nombre total de colonnes, la colonne est ajoutée après la dernière colonne.

Passez dans le paramètre nomCol le nom d'objet de la colonne insérée.

Le paramètre formule peut contenir toute expression valide, soit :
• une instruction,
• une formule générée à l'aide de l'éditeur de formules,
• un appel à une commande 4D,
• un appel à une méthode projet.

Au moment de l'appel de la commande, la formule est analysée puis exécutée.

Note : Utilisez la commande Nom commande afin de définir des formules indépendantes de la langue de l'application (lorsqu'elles font appel à des commandes 4D).

Le paramètre typeDonnées permet de désigner le type des données issues de l'exécution de la formule. Vous devez passer dans ce paramètre une des constantes du thème "Types champs et variables" suivantes :

Constante	Type	Valeur
Est un numérique	Entier long	1
Est un texte	Entier long	2
Est une image	Entier long	3
Est une date	Entier long	4
Est un booléen	Entier long	6
Est une heure	Entier long	11

Si le résultat de la formule ne correspond pas au type de données attendu, une erreur est générée.

Passez dans les paramètres nomEntête et variableEntête le nom d'objet et la variable de l'entête de la colonne insérée.

Note : Les noms d'objets doivent être uniques dans un formulaire. Vous devez veiller à ce que les noms passés dans les paramètres nomCol et nomEntête ne soient pas déjà utilisés. Sinon, la colonne n'est pas créée et une erreur est générée.

Exemple

Nous souhaitons ajouter une nouvelle colonne à la droite de la list box qui contiendra une formule calculant l'âge de l'employé :

```
vAge:="Date du jour-[Employés]DateNaissance)\365"
$der:=Lire nombre colonnes listbox(*;"ListBox1")+1
INSERER COLONNE FORMULE LISTBOX(*;"ListBox1";$der;"ColFormule";vAge;
                                                Est un numérique;"Age";VarEntete)
```

Référence

INSERER COLONNE LISTBOX.

INSERER COLONNE LISTBOX

version 11 (Modifiée)

INSERER COLONNE LISTBOX ({*; }objet; positionCol; nomCol; variableCol; nomEntête; variableEntête)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionCol	Numérique	→	Emplacement de la colonne à insérer
nomCol	Chaîne	→	Nom d'objet de la colonne
variableCol	Tab | Champ | Var	→	Nom de la variable tableau de la colonne ou champ ou variable
nomEntête	Chaîne	→	Nom d'objet de l'en-tête de la colonne
variableEntête	Var Entier	→	Variable d'en-tête de la colonne

Description

La commande INSERER COLONNE LISTBOX insère une colonne dans la list box désignée par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La nouvelle colonne est insérée juste avant la colonne désignée par le paramètre positionCol. Si le paramètre positionCol est supérieur au nombre total de colonnes, la colonne est ajoutée après la dernière colonne.

Passez dans les paramètres nomCol et variableCol le nom d'objet et la variable de la colonne insérée.
• Dans le cadre d'une list box de type tableau, le nom de la variable correspond au nom du tableau dont le contenu sera affiché dans la colonne.
• Dans le cadre d'une list box de type sélection, vous pouvez passer un champ ou une variable dans le paramètre variableCol. Le contenu de la colonne sera alors la valeur du champ ou de la variable, évaluée pour chaque enregistrement de la sélection associée à la list box. Ce type de contenu ne peut être utilisé que lorsque la propriété "Source de données" de la list box est Sélection courante ou Sélection temporaire (cf. section Gestion programmée des objets de type List box). Vous pouvez utiliser des champs ou des variables de type chaîne, numérique, Date, Heure, Image et Booléen.

Dans le contexte de list box basées sur des sélections, INSERER COLONNE LISTBOX permet d'insérer des éléments simples (champs ou variables). Si vous souhaitez manipuler des expressions plus complexes (telles que des formules ou des méthodes), vous devez utiliser la commande INSERER COLONNE FORMULE LISTBOX.

Note : Il n'est pas possible de combiner dans une même list box des colonnes de type tableau (source de données tableaux) et des colonnes de type champ ou variable (source de données sélection).

Passez dans les paramètres nomEntête et variableEntête le nom d'objet et la variable de l'entête de la colonne insérée.

Note : Les noms d'objets doivent être uniques dans un formulaire. Vous devez veiller à ce que les noms passés dans les paramètres nomCol et nomEntête ne soient pas déjà utilisés. Sinon, la colonne n'est pas créée et une erreur est générée.

Exemples

(1) Nous souhaitons ajouter une colonne à la fin de la list box :

```
C_ENTIER LONG(NomVarHeader;$Der;$NbEnr)
TOUT SELECTIONNER([Table 1])
$NbEnr:=Enregistrements dans table([Table 1])
TABLEAU IMAGE(tabImage;$NbEnr)

$Der:=Lire nombre colonnes listbox(*;"ListBox1")+1
INSERER COLONNE LISTBOX(*;"ListBox1";$Der;"ColumnPicture";tabImage;
                                                    "HeaderPicture";NomVarHeader)
```

(2) Nous souhaitons ajouter une colonne à la droite de la list box et lui associer les valeurs du champ [Envois]Frais :

```
$der:=Lire nombre colonnes listbox(*;"ListBox1")+1
INSERER COLONNE LISTBOX(*;"ListBox1";$der;"ColChamp";[Envois]Frais;"NomEntete";
                                                                    VarEntete)
```

Référence

INSERER COLONNE FORMULE LISTBOX, SUPPRIMER COLONNE LISTBOX.

INSERER LIGNE LISTBOX

version 2004

INSERER LIGNE LISTBOX ({*; }objet; positionLigne)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionLigne	Entier long	→	Emplacement de la ligne à insérer

Description

La commande INSERER LIGNE LISTBOX insère une nouvelle ligne dans l'objet list box désigné par les paramètres objet et *.

Note : Cette commande fonctionne uniquement avec les list box basées surdes tableaux. Lorsque cette commande est utilisée avec une list box basée sur des sélections, elle ne fait rien et la variable système *OK* retourne 0.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La ligne est insérée à l'emplacement défini par le paramètre positionLigne. Une nouvelle ligne est automatiquement ajoutée à cet emplacement dans tous les tableaux composant la list box, quel que soit leur type, y compris dans les tableaux (colonnes) masqués.

Si le paramètre positionLigne est supérieur au nombre de lignes des tableaux de la list box, la ligne est ajoutée à la fin de chaque tableau. S'il est égal à 0, la ligne est ajoutée au début de chaque tableau. S'il contient une valeur négative, la commande ne fait rien.

Référence

SUPPRIMER LIGNE LISTBOX.

Lire hauteur lignes listbox

Lire hauteur lignes listbox ({*; }objet) → Entier

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier	←	Hauteur de ligne en pixels

Description

La commande Lire hauteur lignes listbox retourne la hauteur courante (en pixels) des lignes de l'objet list box désigné par les paramètres objet et *. La valeur retournée correspond à la hauteur d'une seule ligne.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Référence

FIXER HAUTEUR LIGNES LISTBOX.

Lire information listbox

version 2004.1 (Modifiée)

Lire information listbox ({*; }objet; info) → Entier long

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
info	Entier long	→	Type d'information à lire
Résultat	Entier long	←	Valeur courante

Description

La commande Lire information listbox permet de récupérer différentes informations concernant la taille et la visibilité des en-têtes et des barres de défilement de l'objet list box désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Passez dans le paramètre info un code correspondant à l'information que vous souhaitez obtenir. Vous pouvez passer une valeur ou l'une des constantes suivantes, placées dans le thème "List box" :

Constante	Type	Valeur	Valeur(s) retournée(s)
Affichage entête listbox	Entier long	0	0=masqué, 1=affiché
Hauteur entête listbox	Entier long	1	Hauteur en pixels
Affichage barre déf hor listbox	Entier long	2	0=masquée, 1=affichée
Hauteur barre déf hor listbox	Entier long	3	Hauteur en pixels
Affichage barre déf ver listbox	Entier long	4	0=masquée, 1=affichée
Largeur barre déf ver listbox	Entier long	5	Largeur en pixels
Position barre déf hor listbox	Entier long	6	Position du curseur en pixels
Position barre déf ver listbox	Entier long	7	Position du curseur en pixels

• Les six premières constantes sont utiles pour calculer la taille de la zone de list box affichée dans le formulaire.

• Lorsque vous utilisez la constante Position barre déf hor listbox ou la constante Position barre déf ver listbox, la commande retourne la position relative du curseur de défilement par rapport à son origine, c'est-à-dire la taille de la partie masquée de la fenêtre, exprimée en pixels. Par défaut, cette position correspond à 0. Combinée par exemple aux informations relatives à la hauteur des lignes, cette valeur permet de connaître le contenu affiché dans la list box.

Exemple

Soit une list box contenant des lignes d'une hauteur de 20 pixels chacune. Vous exécutez l'instruction suivante :

```
$déf:=Lire information listbox(*;"Listbox";Position barre déf ver listbox)
```

Si, par exemple, *$déf* retourne 200, vous pouvez en déduire que la 11e ligne est actuellement la première affichée dans la list box (200/20=10, donc 10 lignes sont masquées).

Référence

CHOIX VISIBLE BARRES DEFILEMENT, MONTRER GRILLE LISTBOX.

Lire largeur colonne listbox

Lire largeur colonne listbox ({*; }objet) → Entier

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier	←	Largeur de colonne en pixels

Description

La commande Lire largeur colonne listbox retourne la largeur (en pixels) de la colonne désignée par les paramètres objet et *. Vous pouvez passer indifféremment une colonne ou un en-tête de colonne de list box dans le paramètre objet.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Référence

FIXER LARGEUR COLONNE LISTBOX.

Lire nombre colonnes listbox

Lire nombre colonnes listbox ({*; }objet) → Entier long

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	←	Nombre de colonnes

Description

La commande Lire nombre colonnes listbox retourne le nombre total de colonnes (visibles ou non) présentes dans la list box désignée par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Référence

SUPPRIMER COLONNE LISTBOX.

Lire nombre lignes listbox

version 11.2 (Modifiée)

Lire nombre lignes listbox ({*; }objet) → Entier long

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Entier long	←	Nombre de lignes

Description

La commande Lire nombre lignes listbox retourne le nombre de lignes présentes dans la list box désignée par les paramètres objet et *.

Note : Lire nombre lignes listbox ne tient pas compte du statut masqué/affiché des lignes. Par exemple, dans une list box contenant 10 lignes et dont les 9 premières sont masquées, Lire nombre lignes listbox retournera 10.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Note : Si les tableaux associés aux colonnes d'un objet List box n'ont pas tous la même taille, seul le nombre d'éléments correspondant au plus petit tableau apparaît dans la list box et est retourné par cette commande.

Référence

INSERER LIGNE LISTBOX, SUPPRIMER LIGNE LISTBOX.

LIRE POSITION CELLULE LISTBOX

version 11.2 (Modifiée)

LIRE POSITION CELLULE LISTBOX ({*; }objet; colonne; ligne{; varCol})

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
colonne	Entier long	←	Numéro de colonne
ligne	Entier long	←	Numéro de ligne
varCol	Pointeur	←	Pointeur sur la variable de colonne

Description

La commande LIRE POSITION CELLULE LISTBOX retourne les numéros de la colonne et de la ligne correspondant à l'emplacement du dernier clic ou de la dernière action de sélection effectuée dans la list box désignée par * et *objet*.
La commande retourne les coordonnées du clic ou de l'action de sélection même lorsque la saisie n'est pas autorisée dans la list box.

Note : Le numéro retourné dans le paramètre ligne ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Si vous passez le paramètre facultatif *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable.

Le paramètre facultatif varCol retourne un pointeur sur la variable (c'est-à-dire le tableau) associée à la colonne.

Cette commande peut être appelée uniquement dans le cadre d'une list box générant l'un des événements formulaire suivants :
- Sur clic et Sur double clic
- Sur avant frappe clavier et Sur après frappe clavier
- Sur après modification
- Sur gain focus et Sur perte focus
- Sur données modifiées
- Sur nouvelle sélection
- Sur avant saisie

Lorsqu'elle est appelée en dehors de ce contexte, LIRE POSITION CELLULE LISTBOX retourne 0 dans colonne et ligne.

Cette commande tient compte des actions de sélection ou de désélection effectuées via la souris, les touches du clavier et la commande EDITER ELEMENT (qui génère l'événement Sur gain focus).

Si la sélection est modifiée via les touches fléchées du clavier, colonne retourne 0. Dans ce cas, s'il est passé, le paramètre varCol retourne Nil.

Les valeurs retournées par la commande ne sont pas mises à jour dans le cas d'un clic droit (ou Control+clic sous Mac OS) sur l'en-tête d'une colonne de la list box.

LIRE TABLE SOURCE LISTBOX ({*; }objet; numTable{; tempo})

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
numTable	Entier long	←	Numéro de la table de la sélection
tempo	Chaîne	←	Nom de la sélection temporaire ou "" pour la sélection courante

Description

La commande LIRE TABLE SOURCE LISTBOX permet de connaître la source courante des données affichées dans la list box désignée par les paramètres * et objet.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La commande retourne dans le paramètre numTable le numéro de la table principale associée à la list box et dans le paramètre facultatif tempo le nom de la sélection temporaire éventuellement utilisée.
Si les lignes de la list box sont liées à la sélection courante de la table, le paramètre tempo, s'il est passé, retourne une chaîne vide. Si les lignes de la list box sont liées à une sélection temporaire, le paramètre tempo retourne le nom de cette sélection temporaire.

Si la list box est associée à des tableaux, numTable retourne -1 et tempo, s'il est passé, retourne une chaîne vide.

Référence

FIXER TABLE SOURCE LISTBOX.

version 11.2 (Modifiée)

LIRE TABLEAUX LISTBOX ({*; }objet; tabNomsCols; tabNomsEntêtes; tabVarCols; tabVarEntêtes; tabColsVisibles; tabStyles)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabNomsCols	Tab alpha	←	Noms d'objet des colonnes
tabNomsEntêtes	Tab alpha	←	Noms d'objet des en-têtes
tabVarCols	Tab pointeur	←	Pointeurs vers les variables des colonnes ou Pointeurs vers les champs des colonnes ou Nil
tabVarEntêtes	Tab pointeur	←	Pointeurs vers les variables des en-têtes
tabColsVisibles	Tab booléen	←	Visibilité de chaque colonne
tabStyles	Tab pointeur	←	Pointeurs vers les tableaux ou les variables de styles de couleurs et de visibilité des lignes ou Nil

Description

La commande LIRE TABLEAUX LISTBOX retourne un ensemble de tableaux synchronisés fournissant diverses informations sur chaque colonne (visible ou non) de la list box désignée par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

A l'issue de l'exécution de la commande :
• Le tableau tabNomsCols contient la liste des noms d'objet de chaque colonne de la list box.
• Le tableau tabNomsEntêtes contient la liste des noms d'objet de chaque en-tête de colonne de la list box.
• Le tableau tabVarCols contient, pour une list box de type tableau, des pointeurs vers les variables (c'est-à-dire les tableaux) associées à chaque colonne de la list box. Pour une list box de type sélection, tabVarCols contient :
- pour une colonne associée à un champ, un pointeur vers le champ associé,
- pour une colonne associée à une variable, un pointeur vers la variable,
- pour une colonne associée à une expression, un pointeur Nil.

• Le tableau tabVarEntêtes contient des pointeurs vers les variables associées à chaque entête de colonne de la list box.
• Le tableau tabColsVisibles contient une valeur booléenne pour chaque colonne, indiquant si la colonne est visible (valeur Vrai) ou masquée (valeur Faux) dans la list box.
• Le tableau tabStyles contient, pour une list box de type tableau, quatre pointeurs vers les quatre tableaux permettant d'appliquer individuellement un style, une couleur de police, une couleur de fond et un statut masqué personnalisés à chaque ligne de la list box. Ces tableaux sont associés à la list box dans la Liste des propriétés en mode Développement. Si un tableau n'est pas spécifié pour la list box, l'élément correspondant de tabStyles contient un pointeur Nil.
Pour une list box de type sélection, tabStyles contient :
- pour chaque paramétrage défini via une variable, un pointeur vers la variable,
- pour chaque paramétrage défini via une expression, un pointeur Nil.

Référence

Lire information listbox.

MONTRER GRILLE LISTBOX

version 2004

MONTRER GRILLE LISTBOX ({*; }objet; horizontal; vertical)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Booléen	→	Vrai=montrer, Faux=cacher
vertical	Booléen	→	Vrai=montrer, Faux=cacher

Description

La commande MONTRER GRILLE LISTBOX permet d'afficher ou de masquer les traits horizontaux et/ou verticaux composant la grille de l'objet list box désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Passez dans les paramètres horizontal et vertical des valeurs booléennes indiquant si les traits correspondants doivent être affichés (Vrai) ou cachés (Faux). Par défaut, la grille est affichée.

Référence

FIXER COULEUR GRILLE LISTBOX, Lire information listbox.

NUMERO COLONNE LISTBOX DEPLACEE

NUMERO COLONNE LISTBOX DEPLACEE ({*; }objet; ancPosition; nouvPosition)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ancPosition	Numérique	←	Ancienne position de la colonne déplacée
nouvPosition	Numérique	←	Nouvelle position de la colonne déplacée

Description

La commande NUMERO COLONNE LISTBOX DEPLACEE retourne dans les paramètres ancPosition et nouvPosition des numéros indiquant respectivement la précédente position et la nouvelle position de la colonne déplacée dans la list box désignée par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Cette commande doit être utilisée en combinaison avec l'événement formulaire Sur déplacement colonne (cf. commande Evenement formulaire).

Note : Cette commande tient compte des colonnes invisibles.

Référence

Evenement formulaire, NUMERO LIGNE LISTBOX DEPLACEE.

NUMERO LIGNE LISTBOX DEPLACEE

version 11.2 (Modifiée)

NUMERO LIGNE LISTBOX DEPLACEE ({*; }objet; ancPosition; nouvPosition)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
ancPosition	Numérique	←	Précédente position de la ligne déplacée
nouvPosition	Numérique	←	Nouvelle position de la ligne déplacée

Description

La commande NUMERO LIGNE LISTBOX DEPLACEE retourne dans les paramètres ancPosition et nouvPosition des numéros indiquant respectivement la précédente position et la nouvelle position de la ligne déplacée dans la list box désignée par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Cette commande doit être utilisée en combinaison avec l'événement formulaire Sur déplacement ligne (cf. commande Evenement formulaire).

Note : Cette commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Référence

Evenement formulaire, NUMERO COLONNE LISTBOX DEPLACEE.

SELECTIONNER LIGNE LISTBOX ({*; }objet; positionLigne{; action})

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionLigne	Entier long	→	Numéro de la ligne à sélectionner
action	Entier long	→	Action de sélection

Description

La commande SELECTIONNER LIGNE LISTBOX provoque la sélection de la ligne de numéro positionLigne dans l'objet list box désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Le paramètre action, s'il est passé, permet de définir l'action de sélection à effectuer lorsqu'une sélection de lignes existe déjà dans la list box. Vous pouvez passer une valeur ou l'une des constantes suivantes, placées dans le thème "List box" :
• Remplacer sélection listbox (0) : la ligne sélectionnée devient la nouvelle sélection et remplace la sélection existante. La commande produit le même effet qu'un clic de l'utilisateur sur une ligne de la list box. Cette action est effectuée par défaut (lorsque le paramètre action n'est pas passé).
• Ajouter à sélection listbox (1) : la ligne sélectionnée est ajoutée à la sélection existante. Si la ligne désignée appartient déjà à la sélection existante, la commande ne fait rien.
• Supprimer de sélection listbox (2) : la ligne sélectionnée est supprimée de la sélection existante. Si la ligne désignée n'appartient pas à la sélection existante, la commande ne fait rien.

Lorsque le paramètre positionLigne ne correspond pas strictement à un numéro de ligne existante, la commande agit de la manière suivante :
• Si positionLigne est <0, la commande ne fait rien, quelle que soit la valeur du paramètre action.
• Si positionLigne vaut 0 et si le paramètre action contient Remplacer sélection listbox ou est omis, toutes les lignes de la listbox sont sélectionnées. Si le paramètre action contient Supprimer de sélection listbox, toutes les lignes de la listbox sont désélectionnées.

• Si la valeur de positionLigne est supérieure au nombre total de lignes contenues dans la listbox, le tableau booléen associé à la listbox est automatiquement redimensionné et l'action de sélection est effectuée. Ce mécanisme permet d'utiliser SELECTIONNER LIGNE LISTBOX avec des commandes "standard" de gestion de tableaux (telles que AJOUTER A TABLEAU) n'entraînant pas de synchronisation immédiate de la listbox.
A l'issue de l'exécution de la méthode, les tableaux sont synchronisés : si le tableau source de la listbox a effectivement été redimensionné, l'action de sélection est effectuée. Sinon, le tableau booléen associé à la listbox reprend sa taille initiale et la commande ne fait rien.

Notes :
• Si vous souhaitez que la list box défile de manière à afficher la ligne nouvellement sélectionnée, utilisez la commande DEFILER LIGNES.
• Pour passer une ligne en mode édition (saisie), utilisez la commande EDITER ELEMENT.
• Si le numéro passé dans positionLigne correspond à une ligne masquée dans la list box, la ligne est sélectionnée mais n'est pas affichée.

Référence

DEFILER LIGNES, EDITER ELEMENT, INSERER LIGNE LISTBOX, SUPPRIMER LIGNE LISTBOX.

SUPPRIMER COLONNE LISTBOX

SUPPRIMER COLONNE LISTBOX ({*; }objet; positionCol{; nombre})

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionCol	Numérique	→	Numéro courant de la colonne à supprimer
nombre	Numérique	→	Nombre de colonnes à supprimer

Description

La commande SUPPRIMER COLONNE LISTBOX supprime une ou plusieurs colonne(s) (visibles ou non) dans la list box désignée par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Si vous ne passez pas le paramètre facultatif nombre, la commande supprime simplement la colonne désignée par le paramètre positionCol.
Sinon, le paramètre nombre indique le nombre de colonnes à supprimer vers la droite à partir de la colonne positionCol (celle-ci incluse).

Si le paramètre positionCol est supérieur au nombre de colonnes de la list box, la commande ne fait rien.

Référence

INSERER COLONNE LISTBOX, Lire nombre colonnes listbox.

SUPPRIMER LIGNE LISTBOX

SUPPRIMER LIGNE LISTBOX ({*; }objet; positionLigne)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
positionLigne	Entier long	→	Numéro de la ligne à supprimer

Description

La commande SUPPRIMER LIGNE LISTBOX supprime la ligne numéro positionLigne (visible ou non) de la list box désignée par les paramètres objet et *.

Note : Cette commande fonctionne uniquement avec les list box basées sur des tableaux. Lorsque cette commande est utilisée avec une list box basée sur des sélections, elle ne fait rien et la variable système *OK* retourne 0.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La ligne positionLigne est supprimée automatiquement de tous les tableaux composant la list box.
Notez qu'après l'exécution de la commande, il n'y a plus d'élément sélectionné dans la list box.

Si le paramètre positionLigne est supérieur au nombre de lignes des tableaux de la list box, la commande ne fait rien.

Note : Cette commande ne tient pas compte de l'éventuel statut masqué/affiché des lignes de la list box.

Référence

INSERER LIGNE LISTBOX, Lire nombre lignes listbox.

TRIER COLONNES LISTBOX ({*; }objet; numCol; sensDuTri{; numCol2; sensDuTri2; ...; numColN; sensDuTriN})

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
numCol	Entier long	→	Numéro(s) de colonne(s) de tri
sensDuTri	> ou <	→	> pour effectuer un tri croissant ou < pour effectuer un tri décroissant

Description

La commande TRIER COLONNES LISTBOX permet de trier (réordonner) toutes les lignes de la list box désignée par les paramètres objet et * sur la base des valeurs d'une ou plusieurs colonne(s).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Passez dans numCol le numéro de la colonne dont les valeurs seront utilisées comme critère de tri. Vous pouvez utiliser tout type de données, à l'exception des colonnes contenant des images et des pointeurs.

Passez dans sensDuTri le symbole > ou < indiquant l'ordre du tri : croissant ou décroissant. Si sensDuTri est égal au symbole "supérieur à" (>), l'ordre du tri est croissant. Si sensDuTri est égal au symbole "inférieur à" (<), l'ordre du tri est décroissant.

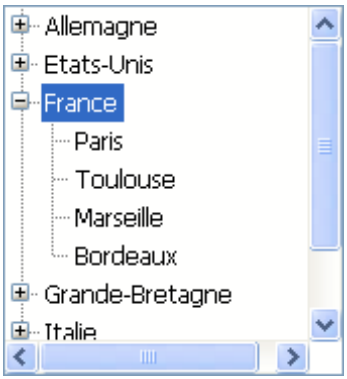
Vous pouvez définir des tris multi-niveaux : pour cela, passez autant de paires numCol;sensDuTri que nécessaire. Le niveau de tri est défini par la position du paramètre lors de l'appel.

Conformément au principe de fonctionnement des list box, les colonnes sont synchronisées, ce qui signifie que le tri d'une colonne est automatiquement répercuté sur toutes les colonnes de l'objet.

37

Listes hiérarchiques

Les listes hiérarchiques sont des objets de formulaire permettant d'afficher des données sous forme de listes comportant un ou plusieurs niveaux qu'il est possible de déployer ou de contracter.



Dans les formulaires, les listes hiérarchiques peuvent servir à l'affichage ou la saisie de données. Chaque élément de liste peut contenir jusqu'à de 2 milliards de caractères (taille maximale d'un champ texte) et être associé à une icône. Les listes hiérarchiques prennent généralement en charge les clics, double-clics, navigation au clavier ou encore le glisser-déposer. Il est possible d'effectuer une recherche parmi le contenu d'une liste (commande Chercher dans liste).

Création et modification

Les listes hiérarchiques peuvent être créées entièrement par programmation (via les commandes Nouvelle liste ou Copier liste) ou à partir d'énumérations définies dans l'éditeur d'énumérations en mode Développement (commande Charger liste).

Le contenu et l'apparence des listes hiérarchiques sont gérés par programmation, à l'aide des commandes du thème "Liste hiérarchique". Certaines caractéristiques d'apparence spécifiques peuvent également être définies via les commandes génériques du thème "Propriétés des objets" (cf. ci-dessous).

RefListe et nom d'objet

Une liste hiérarchique est à la fois un **objet de langage** existant en mémoire et un **objet de formulaire**.
L'**objet de langage** est référencé de manière unique par un identifiant interne, de type Entier long, désigné par RefListe dans ce manuel. Cet identifiant est retourné par les commandes permettant de créer des listes Nouvelle liste, Copier liste, Charger liste, BLOB vers liste. Il n'existe qu'une seule instance en mémoire de l'objet de langage et toute modification effectuée sur cet objet est immédiatement répercutée dans tous les endroits où il est utilisé.
L'**objet de formulaire** n'est pas nécessairement unique : il peut exister plusieurs représentations d'une même liste hiérarchique dans un même formulaire ou dans des formulaires différents. Comme pour les autres objets de formulaire, vous désignez l'objet dans le langage via la syntaxe (*;"NomListe"...).

Vous connectez l'"objet de langage" liste hiérarchique avec l'"objet de formulaire" liste hiérarchique par l'intermédiaire de la variable contenant la valeur de l'identifiant unique RefListe. Par exemple, si vous écrivez :

```
maliste:=Nouvelle liste
```

... il suffit d'associer le nom de variable *maliste* à l'objet de formulaire Liste hiérarchique dans la liste des propriétés afin qu'il gère l'objet de langage dont la RefListe est stockée dans *maListe*.

Chaque représentation de liste dispose de caractéristiques propres et partage des caractéristiques communes avec l'ensemble des représentations. Les caractéristiques propres à chaque représentation de liste sont les suivantes :
• la sélection,
• l'état déployé/contracté des éléments,
• la position du curseur de défilement.
Les autres caractéristiques (police, style, filtre de saisie, couleur, contenu de la liste, icônes, etc.) sont communes à toutes les représentations et ne peuvent pas être modifiées séparément.
Par conséquent, lorsque vous utilisez des commandes se basant sur la configuration déployé/contracté ou l'élément courant, par exemple Nombre elements (lorsque le paramètre * final n'est pas passé), il importe de pouvoir désigner sans ambiguïté la représentation à utiliser.

Vous devez utiliser l'identifiant de type RefListe avec les commandes du langage lorsque vous souhaitez désigner la liste hiérarchique résidant en mémoire.
Si vous souhaitez désigner la représentation au niveau du formulaire d'un objet Liste hiérarchique, vous devez utiliser le nom de l'objet (type chaîne) dans la commande, via la syntaxe (*;"NomListe"...). Cette syntaxe est identique à celle en vigueur dans les commandes du thème "Propriétés des objets". Elle est acceptée par la plupart des commandes du thème "Liste hiérarchique" agissant sur les propriétés des listes (reportez-vous à la description des commandes du thème).

Attention, dans le cas des commandes définissant des propriétés, la syntaxe basée sur le nom d'objet ne signifie pas que seul l'objet de formulaire désigné sera modifié par la commande, mais que l'action de la commande sera basée sur l'état de cet objet. Les caractérisques communes des listes hiérarchiques sont toujours modifiées dans toutes les représentations.
Par exemple, si vous passez l'instruction FIXER POLICE ELEMENT(*;"maliste1";*;lapolice), vous indiquez que vous souhaitez modifier la police d'un élément de la liste hiérarchique associée à l'objet de formulaire maliste1. La commande tiendra compte de l'élément courant de l'objet maliste1 pour définir l'élément à modifier, mais cette modification sera reportée dans toutes les représentations de la liste dans tous les process.

Prise en compte du @
Comme pour les autres commandes de gestion des propriété d'objets, il est possible d'utiliser le caractère "@" dans le paramètre NomListe. En principe, cette syntaxe permet de désigner un ensemble d'objets dans le formulaire. Toutefois, dans le contexte des commandes de liste hiérarchique, ce principe n'est pas applicable dans tous les cas. Cette syntaxe aura deux effets différents en fonction du type de commande :
• Pour les commandes fixant des propriétés, cette syntaxe désigne tous les objets dont le nom correspond (fonctionnement standard). Par exemple, le paramètre "LH@" désigne tous les objets de type liste hiérarchique dont le nom débute par "LH". Ces commandes sont :
CHANGER ELEMENT
CHANGER PROPRIETES ELEMENT
FIXER ICONE ELEMENT
FIXER PARAMETRE ELEMENT
FIXER POLICE ELEMENT
INSERER DANS LISTE
SELECTIONNER ELEMENTS PAR POSITION
SUPPRIMER DANS LISTE
• Pour les commandes récupérant des propriétés, cette syntaxe désigne le premier objet dont le nom correspond. Ces commandes sont :
Chercher dans liste
Element parent
Elements selectionnes
INFORMATION ELEMENT
LIRE ICONE ELEMENT
LIRE PARAMETRE ELEMENT
Lire police element
LIRE PROPRIETES ELEMENT
Nombre elements
Position element liste

Commandes génériques utilisables avec les listes hiérarchiques

Il est possible de modifier l'apparence d'une liste hiérarchique dans un formulaire à l'aide de plusieurs commandes 4D génériques. Vous devez passer à ces commandes soit le nom d'objet de la liste hiérarchique (en utilisant le paramètre *), soit son nom de variable (syntaxe standard).

Note : Dans le cas des listes hiérarchiques, la variable de formulaire contient la valeur de RéfListe. Si vous exécutez une commande de modification d'attribut en lui passant la variable associée à la liste hiérarchique, il ne sera pas possible de définir la liste cible en cas de multi-représentation. Seul le nom d'objet permet donc de différencier individuellement chaque représentation.

Voici la liste des commandes utilisables avec les listes hiérarchiques. Hormis DEFILER LIGNES, toutes ces commandes appartiennent au thème "Propriétés des objets" :
CHANGER JEU DE CARACTERES
CHANGER STYLE
CHANGER TAILLE
CHOIX COULEUR
CHOIX FILTRE SAISIE
CHOIX SAISISSABLE
CHOIX VISIBLE BARRES DEFILEMENT
DEFILER LIGNES (thème "Interface utilisateur")
FIXER COULEURS RVB

Rappel : A l'exception de la commande DEFILER LIGNES, ces commandes modifient toutes les représentations d'une même liste, même si vous désignez une liste via son nom d'objet.

Priorité des commandes de propriété

Certaines propriétés d'une liste hiérarchique (par exemple l'attribut saisissable ou la couleur) peuvent être définies de trois manières : via la Liste des propriétés en mode Développement, via une commande du thème "Propriétés des objets" ou via une commande du thème "Liste hiérarchique".
Lorsque ces trois moyens sont utilisés pour définir les propriétés d'une liste, l'ordre de priorité suivant est appliqué :
1. Commandes du thème "Liste hiérarchique"
2. Commandes générique de propriété d'objet
3. Paramètres de la Liste des propriétés
Ce principe est appliqué quel que soit l'ordre d'appel des commandes. Si une propriété d'élément est modifiée individuellement via une commande de liste hiérarchique, la commande de propriété d'objet équivalente sera sans effet sur cet élément même si elle est appelée ultérieurement. Par exemple, si vous modifiez la couleur d'un élément via la commande CHANGER PROPRIETES ELEMENT, la commande CHOIX COULEUR n'aura aucun effet sur cet élément.

Gestion des éléments par position ou par référence

Vous pouvez généralement travailler de deux manières avec le contenu des listes hiérarchiques : par position ou par référence.
• Lorsque vous travaillez par position, 4D se base sur la position relative des éléments dans la liste affichée à l'écran pour les identifier. Le résultat sera différent selon que certains éléments hiérarchiques sont déployés ou non. A noter qu'en cas de multi-représentation, chaque objet de formulaire comporte sa propre configuration d'éléments contractés/déployés.
• Lorsque vous travaillez par référence, 4D se base sur le numéro unique réfElément des éléments de la liste. Chaque élément peut être ainsi désigné, quelle que soit sa position ou son affichage dans la liste hiérarchique.

Exploiter les numéros de référence des éléments (réfElément)

Chaque élément d'une liste hiérarchique dispose d'un numéro de référence (réfElément) de type Entier long. Cette valeur est destinée uniquement à votre propre usage : 4D ne fait que la maintenir.

Attention : Vous pouvez utiliser comme numéro de référence toute valeur de type entier long, sauf la valeur 0. En effet, pour la plupart des commandes de ce thème, la valeur 0 permet de désigner le dernier élément ajouté à la liste.

Voici quelques astuces quant à l'utilisation du numéro de référence unique :

(1) Vous n'avez pas besoin d'identifier chaque élément de façon unique (niveau débutant).
• Premier exemple : vous construisez par programmation un système d'onglets, par exemple, un carnet d'adresses. Comme le système vous retournera le numéro de l'onglet sélectionné, vous n'aurez probablement pas besoin de davantage d'informations. Dans ce cas, ne vous préoccupez pas des numéros de référence des éléments : passez n'importe quelle valeur (hormis 0) dans le paramètre réfElément. Notez que pour un système de carnet d'adresses, vous pouvez prédéfinir une liste A, B,..., Z en mode Développement. Vous pouvez également la créer par programmation afin d'éliminer les lettres pour lesquelles il n'y a pas d'enregistrement.
• Deuxième exemple : en travaillant avec une base, vous construisez progressivement une liste de mots-clés. Vous pouvez sauvegarder la liste à la fin de chaque session, en utilisant les commandes STOCKER LISTE ou LISTE VERS BLOB, et la recharger au début de chaque session, à l'aide des commandes Charger liste ou BLOB vers liste. Vous pouvez afficher cette liste dans une palette flottante ; lorsque l'utilisateur clique sur un mot-clé de la liste, l'élément choisi est inséré dans la zone saisissable sélectionnée du process de premier plan. Vous pouvez également utiliser le glisser-déposer. En tout état de cause, l'important est que vous ne traitez que l'élément sélectionné (par clic ou glisser-déposer), car les commandes Elements selectionnes (en cas de clic) et PROPRIETES GLISSER DEPOSER vous retournent la position de l'élément que vous devez traiter. En utilisant cette valeur de position, vous obtenez le libellé de l'élément grâce à la commande INFORMATION ELEMENT. Ici aussi, vous n'avez pas besoin d'identifier de façon unique chaque élément ; vous pouvez passer n'importe quelle valeur (hormis 0) dans le paramètre réfElément.

(2) Vous avez besoin d'identifier partiellement les éléments de la liste (niveau intermédiaire). Vous utilisez le numéro de référence de l'élément pour stocker l'information nécessaire lorsque vous devez agir sur un élément ; ce point est détaillé dans l'exemple de la commande AJOUTER A LISTE. Dans cet exemple, nous utilisons les numéros de référence des éléments pour stocker des numéros d'enregistrements. Cependant, nous devons pouvoir établir une distinction entre les éléments qui correspondent aux enregistrements [Départements] et ceux qui correspondent aux enregistrements [Employés].

(3) Vous avez besoin d'identifier les éléments de la liste de façon unique (niveau avancé). Vous programmez une gestion élaborée de listes hiérarchiques, dans laquelle vous devez absolument pouvoir identifier chaque élément de manière unique à tous les niveaux de la liste. Un moyen simple d'implémenter ce fonctionnement est de maintenir un compteur personnel. Supposons que vous créez une liste *hlList* à l'aide de la commande Nouvelle liste. A ce stade, vous initialisez un compteur *vlhCounter* à 1. A chaque fois que vous appelez AJOUTER A LISTE ou INSERER DANS LISTE, vous incrémentez ce compteur (vlhCounter:=vlhCounter+1), et vous passez le compteur comme numéro de référence de l'élément. L'astuce consiste à ne pas décrémenter le compteur lorsque vous détruisez des éléments — le compteur ne peut qu'augmenter. En procédant ainsi, vous garantissez l'unicité des numéros de référence des éléments. Puisque ces numéros sont des valeurs de type Entier long, vous pouvez ajouter ou insérer plus de deux milliards d'éléments dans une liste qui a été réinitialisée... (si vous manipulez d'aussi grandes quantités d'éléments, cela signifie généralement que vous devriez utiliser une table plutôt qu'une liste.)

Note : Si vous exploitez les Opérateurs sur les bits, vous pouvez également utiliser les numéros de référence des éléments pour stocker des informations qui peuvent être logées dans un Entier long, c'est-à-dire 2 Entiers, des valeurs de 4 octets ou encore 32 Booléens.

Quand avez-vous besoin de numéros de référence uniques ?
Dans la plupart des cas, lorsque vous utilisez des listes hiérarchiques pour des besoins d'interface utilisateur, pour lesquels seul l'élément sélectionné (par un clic ou par glisser-déposer) est important, vous n'avez pas besoin d'utiliser les numéros de référence des éléments. Les commandes Elements selectionnes et INFORMATION ELEMENT vous fournissent toutes les informations nécessaires à la gestion de l'élément sélectionné. De plus, des commandes telles que INSERER DANS LISTE ou SUPPRIMER DANS LISTE vous permettent de manipuler la liste de manière "relative" à l'élément sélectionné.
En pratique, vous devez vous préoccuper des numéros de référence d'éléments lorsque vous voulez accéder directement par programmation à n'importe quel élément de la liste, et pas nécessairement à l'élément couramment sélectionné.

AJOUTER A LISTE

version 11 (Modifiée)

AJOUTER A LISTE (liste; texteElément; réfElément{; sous_Liste; déployée})

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de liste
texteElément	Chaîne	→	Libellé du nouvel élément
réfElément	RéfElément	→	Numéro de référence unique du nouvel élément
sous_Liste	RéfListe	→	Sous-liste optionnelle à rattacher au nouvel élément
déployée	Booléen	→	Indique si la sous-liste doit être déployée ou non

Description

La commande AJOUTER A LISTE ajoute un nouvel élément à la liste hiérarchique dont vous avez passé le numéro de référence dans le paramètre liste.

Vous passez le libellé de l'élément dans le paramètre texteElément. Vous pouvez passer une expression de type Alpha ou Texte, pouvant contenir jusqu'à 2 milliards de caractères.

Vous passez le numéro de référence unique de l'élément (de type Entier long) dans le paramètre réfElément. Même si nous qualifions ce numéro de référence d'élément comme unique, vous pouvez en réalité passer la valeur que vous voulez. Reportez-vous à la section Gestion des listes hiérarchiques pour plus d'informations sur le paramètre réfElément.

Si vous souhaitez également que l'élément comporte des sous-éléments, passez un numéro de référence de liste valide dans le paramètre sous_Liste. Dans ce cas, vous devez également passer le paramètre déployé. Passez Vrai ou Faux dans ce paramètre pour que cette sous-liste s'affiche respectivement déployée ou contractée.

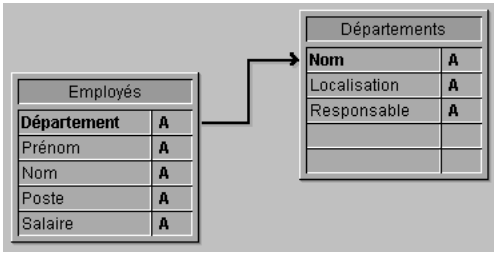
La référence de la liste que vous passez dans sous_Liste doit être une liste existante. Elle peut comporter un seul niveau ou contenir elle-même des sous-listes. Si vous ne voulez pas rattacher de sous-liste au nouvel élément, omettez le paramètre ou passez 0. Si vous passez le paramètre sous_Liste et ne passez pas le paramètre déployée, la sous-liste apparaît par défaut contractée.

Conseils

• Pour insérer un nouvel élément dans une liste, utilisez INSERER DANS LISTE. Pour modifier le libellé d'un élément existant ou sa sous-liste, ainsi que son état déployé/contracté, utilisez CHANGER ELEMENT.
• Pour changer l'apparence de l'élément ajouté, utilisez CHANGER PROPRIETES ELEMENT.

Exemple

Voici une vue partielle de la structure d'une base :



Les tables [Départements] et [Employés] contiennent les enregistrements suivants :

Départements

Nom	Localisation	Responsable
Biologie marine	Vivarium 11	Robert Ibéri
Comptabilité	2ème étage	Louis Berouet
Ventes	RDC ouest	Eliane Bergis

Employés

Département	Prénom	Nom
Biologie marine	Daphné	Vaudelles
Comptabilité	Gérard	Périet
Ventes	Guy	Chartret
Comptabilité	Gilbert	Darieux
Biologie marine	Arnaud	Schmitt
Ventes	Frédérique	Dalhoum
Biologie marine	Pierre	Marty

Vous voulez utiliser une liste hiérarchique, appelée hlList, qui affiche les départements, et pour chaque département, une sous-liste contenant les employés travaillant dans ce département.

La méthode objet de hlList est la suivante:

```
      ` Méthode objet Liste hiérarchique hlList

   Au cas ou

      : (Evenement formulaire=Sur chargement)
         C_ENTIER LONG(hlList;$hSousListe;$vlDépartement;$vlEmployé)
            ` Créer une nouvelle liste hiérarchique vide
         hlList:=Nouvelle liste
            ` Sélectionner tous les enregistrements de la table [Départements]
         TOUT SELECTIONNER([Départements])
            ` Pour chaque Département
         Boucle ($vlDepartement;1;Enregistrements trouves([Départements]))
                  ` Sélectionner les employés de ce département
            LIEN RETOUR([Départements]Nom)
               ` Combien sont-ils?
            $vlNbEmployés:=Enregistrements trouves([Employés])
               ` Y a-t-il au moins un employé dans ce département?
            Si ($vlNbEmployés>0)
                     ` Créer une sous-liste pour l'élément Département
               $hSousListe:=Nouvelle liste
                  ` Pour chaque Employé
               Boucle ($vlEmployé;1;Enregistrements trouves([Employés]))
                        ` Ajouter l'élément Employé à la sous-liste
                        ` Noter que le numéro de l'enregistrement [Employés]
                        ` est passé comme numéro de référence de l'élément
                  AJOUTER A LISTE($hSousListe;[Employés]Nom+", "+[Employés]Prénom;
                                                   Numero enregistrement([Employés]))
                        ` Aller à l'enregistrement [Employés] suivant
                  ENREGISTREMENT SUIVANT([Employés])
               Fin de boucle
            Sinon
                     ` Pas d'Employé, pas de sous-liste pour l'élément Département
               $hSousListe:=0
            Fin de si
```

```
            ` Ajouter l'élément Département à la liste principale
            ` Notez que le numéro de l'enregistrement [Départements]
            ` est passé comme numéro de référence de l'élément. Le bit #31du numéro
            ` de référence de l'élément est forcé à 1. Ainsi nous pourrons faire la
            ` distinction entre les éléments Département et Employés (cf. note)
        AJOUTER A LISTE(hlList;[Départements]Nom;
                                0x80000000 | Numero enregistrement([Départements]);
                                                $hSousListe;$hSousListe#0)
            ` Passer l'élément Département en gras pour renforcer la hiérarchie de la
            ` liste
        CHANGER PROPRIETES ELEMENT(hlList;0;Faux;Gras;0)
            ` Aller au département suivant
        ENREGISTREMENT SUIVANT([Départements])
    Fin de boucle
        ` Trier toute la liste en ordre croissant
    TRIER LISTE(hlList;>)
        ` Afficher la liste en style Windows et forcer la hauteur de ligne mini à 14 Pts
    CHANGER PROPRIETES LISTE(hlList;A la Windows;Réf icône Windows;14)

: (Evenement formulaire=Sur libération)
        ` La liste n'est plus utile. N'oubliez pas de l'effacer !
    SUPPRIMER LISTE(hlList;*)

: (Evenement formulaire=Sur double clic)
        ` Il y a eu un double-clic
        ` Obtenir la position de l'élément sélectionné
    $vlÉlémentPos:=Elements selectionnes(hlList)
        ` A toutes fins utiles, vérifier la position
    Si ($vlÉlémentPos # 0)
            ` Obtenir l'information de l'élément de la liste
        INFORMATION ELEMENT(hlList;$vlÉlémentPos;$vlÉlémentRef;$vsÉlémentText;
                                            $vlÉlémentSousListe;$vbÉlémentDéployé)
            ` Cet élément est-il l'élément d'un Département?
        Si ($vlÉlémentRef ?? 31)
                ` Si oui, c'est un double-clic sur un élément Département
            ALERTE("Vous avez double-cliqué sur l'élément Département "+
                                        Caractere(34)+$vsÉlémentText+Caractere(34)+".")
        Sinon
            ` Sinon, c'est un double-clic sur un élément Employé. Avec le numéro
            ` de référence de l'élément parent, trouver l'enregistrement [Départements]
```

```
                        ALLER A ENREGISTREMENT([Départements];Element parent(hlList;
                                                                    $vlÉlémentRef)?-31)
                    ` Signaler où l'Employé travaille et de qui il dépend
                ALERTE("Vous avez double-cliqué sur l'élément Employé "+Caractere(34)+
                    $vsÉlémentText+Caractere(34)+" qui travaille dans le Département "+
                                        Caractere(34)+[Départements]Nom+Caractere(34)+
                    " dont le responsable est "+Caractere(34)+[Départements]Responsable+
                                                        Caractere(34)+".")
            Fin de si
        Fin de si

    Fin de cas

        ` Note : 4D peut stocker jusqu'à 1 milliard d'enregistrements par table. Le numéro
        ` d'enregistrement tient sur 24 bits. Dans notre exemple, nous utilisons le bit #31 de
        ` l'octet supérieur inutilisé pour différencier les éléments des Employés des
        ` Départements.
```

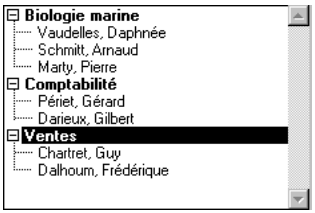
Dans cet exemple, il y a une seule raison d'établir une distinction entre les éléments Départements et les éléments Employés :

1. Nous stockons les numéros d'enregistrements dans les numéros de référence des éléments. En conséquence, nous avons toutes les chances de rencontrer des éléments Départements dont les numéros de référence sont les mêmes que ceux des éléments Employés.

2. Nous utilisons la commande Element parent pour récupérer le parent de l'élément sélectionné. Si nous cliquons sur un élément Employés dont le numéro d'enregistrement associé est 10, et s'il existe aussi un élément Départements qui a le numéro 10, l'élément Départements sera trouvé en premier par Element parent quand cette fonction passera la liste en revue pour repérer l'élément avec le numéro de référence que nous passons. La commande retournera le parent de l'élément Départements et non celui de l'élément Employés.

C'est pourquoi nous avons choisi des numéros de référence d'éléments uniques, non pas pour des questions de principe, mais parce que nous devions différencier les éléments de Départements et d'Employés.

Dans le formulaire en exécution, la liste apparaîtra ainsi :



Note : Cet exemple est utile dans le cadre de la gestion de l'interface utilisateur, si vous manipulez un nombre limité d'enregistrements. Souvenez-vous que les listes sont conservées en mémoire ; donc, ne construisez pas d'interfaces utilisateur exploitant des listes hiérarchiques comportant des millions d'éléments.

Référence

CHANGER ELEMENT, CHANGER PROPRIETES ELEMENT, INSERER DANS LISTE.

CHANGER ELEMENT ({*; } liste; réfElément | *; textElément; nouvelRéf{; sous_Liste; déployée})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
textElément	Alpha	→	Nouveau libellé d'élément
nouvelRéf	Entier long	→	Nouveau numéro de référence d'élément
sous_Liste	RéfListe	→	Nouvelle sous-liste rattachée à l'élément ou 0 = pas de sous-liste (détacher sous-liste courante) ou -1 = pas de changement
déployée	Booléen	→	Indique si la sous-liste doit être déployée/contractée

Description

La commande CHANGER ELEMENT modifie l'élément désigné par le paramètre réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.
Vous pouvez également passer 0 dans réfElément afin de désigner le dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE).
Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la section Gestion des listes hiérarchiques.

Vous pouvez passer le nouveau libellé de l'élément dans le paramètre textElément. Si vous souhaitez changer le numéro de référence de l'élément, passez la nouvelle valeur dans le paramètre nouvelRéf, sinon passez la même valeur que dans réfElément.

Si vous voulez associer une sous-liste à l'élément, passez le numéro de référence de la sous-liste dans le paramètre sous_Liste. Dans ce cas, vous devez également spécifier si la nouvelle sous-liste devra apparaître déployée ou contractée en passant respectivement Vrai ou Faux dans le paramètre déployée.

Si vous voulez dissocier de l'élément une sous-liste qui lui est actuellement rattachée, passez 0 (zéro) dans sous_Liste. Dans ce cas, il est conseillé d'avoir préalablement obtenu le numéro de référence de cette liste à l'aide de la commande INFORMATION ELEMENT, afin de pouvoir effacer la sous-liste avec la commande SUPPRIMER LISTE si vous n'en avez plus besoin.

Si vous ne souhaitez pas modifier les propriétés de sous-liste de l'élément, passez -1 dans le paramètre sous_Liste.

Exemples

(1) Nous supposons que hList est une liste dont les éléments ont des numéros de référence uniques. La méthode objet suivante d'un bouton ajoute une sous-liste à l'élément actuellement sélectionné dans la liste hList :

```
$vlItemPos:=Elements selectionnes(hList)
Si ($vlItemPos>0)
    INFORMATION ELEMENT(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSouslist;
                                                                $vbExpanded)
    $vbNouvSousList:=Non(Liste existante($hSouslist))
    Si ($vbNouvSousList)
        $hSouslist:=Nouvelle liste
    Fin de si
    vlUniqueRef:=vlUniqueRef+1
    AJOUTER A LISTE($hSousList;"Nouvel élément";vlUniqueRef)
    Si ($vbNouvSousList)
        CHANGER ELEMENT(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSouslist;Vrai)
    Fin de si
    SELECTIONNER ELEMENTS PAR REFERENCE(hList;vlUniqueRef)
Fin de si
```

(2) Reportez-vous à l'exemple de la commande INFORMATION ELEMENT.
(3) Reportez-vous à l'exemple de la commande AJOUTER A LISTE.

Référence

CHANGER PROPRIETES ELEMENT, INFORMATION ELEMENT, LIRE PROPRIETES ELEMENT.

CHANGER PROPRIETES ELEMENT ({*; }liste; réfElément | *; saisissable; styles; icône{; couleur})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
saisissable	Booléen	→	Vrai = Saisissable, Faux = Non-saisissable
styles	Numérique	→	Style de police pour l'élément
icône	Numérique	→	Numéro de ressource Mac OS 'cicn' ou 65536 + numéro de ressource Mac OS 'PICT' ou 131072 + numéro de référence d'image
couleur	Entier long	→	Valeur de couleur RVB ou -1 = rétablir couleur originale

Description

La commande CHANGER PROPRIETES ELEMENT modifie l'élément désigné par le paramètre réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.
Vous pouvez également passer 0 dans réfElément afin de demander la modification du dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE).

Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la section Gestion des listes hiérarchiques.

Note : Pour changer le libellé d'un élément ou de ses sous-listes, utilisez la commande CHANGER ELEMENT.

Si vous souhaitez que l'élément soit saisissable, passez Vrai dans le paramètre saisissable, sinon passez Faux.

Important : Pour qu'un élément soit saisissable, il doit appartenir à une liste elle-même saisissable. Pour déclarer une liste saisissable, utilisez la commande CHOIX SAISISSABLE. La commande CHANGER PROPRIETES ELEMENT vous permet de déclarer un élément individuel saisissable ou non. La modification de la propriété saisissable au niveau de la liste ne change pas la propriété saisissable individuelle de chaque élément. Un élément ne peut être saisissable que si la liste **et** l'élément le sont.

Vous pouvez définir le style de l'élément dans le paramètre styles. Vous passez une ou une combinaison des constantes prédéfinies suivantes (thème Styles de caractères) :

Constante	Type	Valeur
Normal	Entier long	0
Gras	Entier long	1
Italique	Entier long	2
Souligné	Entier long	4

Si vous souhaitez associer une icône à l'élément, passez une des valeurs numériques suivantes dans le paramètre icône :
• N, où N est le numéro d'une ressource Mac OS 'cicn'
• Utiliser ressource PICT+N, où N est le numéro d'une ressource Mac OS 'PICT'.
• Utiliser réf image+N, où N est le numéro de référence d'une image stockée dans la bibliothèque d'images de 4D, en mode Développement.
• Si vous ne souhaitez pas associer d'image à l'élément, passez 0 (zéro) dans icône.

Notes :
• Utiliser ressource PICT et Utiliser réf image sont des constantes prédéfinies placées dans le thème Listes hiérarchiques.

• Si vous souhaitez utiliser des expressions image 4D (champs, variables...) pour définir les icônes des éléments, utilisez la commande FIXER ICONE ELEMENT.

Le paramètre couleur (facultatif) permet de modifier la couleur du texte de l'élément. La couleur doit être définie sous forme de couleur RVB, c'est-à-dire un entier long de 4 octets au format 0x00RRVVBB. Pour plus d'informations sur ce format, reportez-vous à la description de la commande FIXER COULEURS RVB. Passez -1 dans le paramètre couleur pour rétablir la couleur d'origine de l'élément.

Exemples

(1) Reportez-vous à l'exemple de la commande AJOUTER A LISTE.

(2) L'exemple suivant passe le texte de l'élément courant de liste en gras et en rouge vif :

```
CHANGER PROPRIETES ELEMENT(liste;*;Vrai;Gras;0;0x00FF0000)
```

Référence

CHANGER ELEMENT, FIXER ICONE ELEMENT, LIRE PROPRIETES ELEMENT.

CHANGER PROPRIETES LISTE

version 2004 (Modifiée)

CHANGER PROPRIETES LISTE (liste; apparence{; icône{; hauteurLigne{; doubleClic{; multiSélection{; modifiable}}}}})

Paramètre	Type		Description
liste	RefListe	→	Numéro de référence de la liste
apparence	Numérique	→	Style graphique de la liste 1 = Liste hiérarchique à la Macintosh 2 = Liste hiérarchique à la Windows 0 = Apparence auto en fonction de la plate-forme
icône	Numérique	→	Référence de ressource Mac OS ‘cicn’ ou 0 = icône par défaut de la plate-forme
hauteurLigne	Numérique	→	Hauteur minimale de la ligne (pixels)
doubleClic	Entier long	→	Déploiement/contraction sur double-clic 0 = autoriser, 1= empêcher
multiSélection	Entier long	→	Sélections multiples 0 = interdire (défaut), 1 = autoriser
modifiable	Entier long	→	Enumération modifiable 0 = non, 1 = oui (défaut)

Description

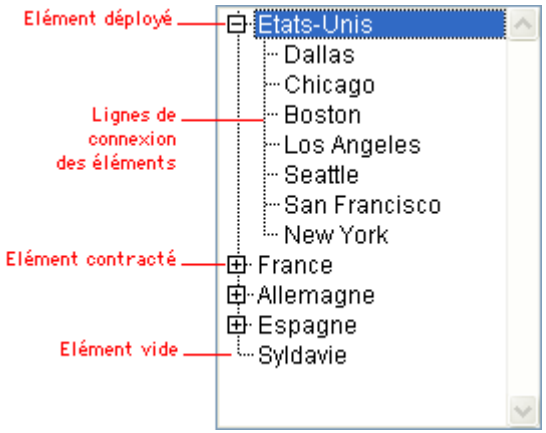
La commande CHANGER PROPRIETES LISTE définit l'apparence et le fonctionnement de la liste hiérarchique dont la référence est passée dans le paramètre liste.

Vous pouvez passer dans le paramètre apparence une des constantes prédéfinies suivantes, fournies par 4D dans le thème Listes hiérarchiques :

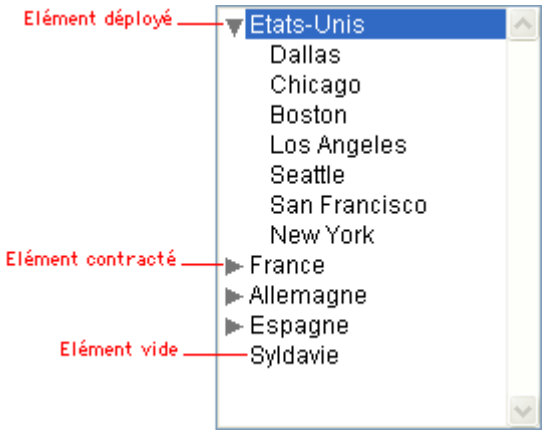
Constante	Type	Valeur
A la Macintosh	Entier long	1
A la Windows	Entier long	2

Avec l'apparence Windows, la liste contient des lignes pointillées qui connectent les éléments et les sous-élements. Une icône "+" signale un élement dont la sous-liste est contractée, une icône "–" signale un élément dont la sous-liste est déployée. Les éléments sans sous-éléments n'ont pas d'icône.

Voici la liste hiérarchique par défaut à la Windows :



Avec l'apparence Macintosh, la liste s'affiche sans lignes pointillées. Une icône en forme de triangle pointant vers la droite signale un élément dont la sous-liste est contractée, une icône en forme de triangle pointant vers le bas signale un élément dont la sous-liste est déployée. Les éléments sans sous-éléments n'ont pas d'icône. Voici la liste hiérarchique par défaut à la Macintosh :



Si vous affichez une liste hiérarchique sans appeler CHANGER PROPRIETES LISTE ou passez 0 dans le paramètre apparence, la liste s'affiche avec l'apparence par défaut, en fonction de la plate-forme sélectionnée en mode Développement dans l'éditeur de formulaires.

Le paramètre icône indique l'icône affichée pour chaque élément. La valeur passée dans icône définit l'icône pour les sous-listes contractées et la valeur icône+1 définit l'icône pour les sous-listes déployées.

Si, par exemple, vous passez 15000 dans icône, l'icône couleur 'cicn' ID=15000 sera affichée pour chaque sous-liste contractée et l'icône couleur 'cicn' ID=15001 sera affichée pour chaque sous-liste déployée.

Dans ce cas, il est important de disposer effectivement de ces deux ressources d'icône couleur 'cicn' dans le fichier de structure de votre base. Si aucune icône couleur n'est présente, les éléments correspondants sont affichés sans icône (à noter que c'est un moyen d'afficher une liste sans icônes).

Attention : Lorsque vous créez des ressources d'icône couleur 'cicn', utilisez des numéros de référence (ID) de ressource supérieurs ou égaux à 15000. Les numéros de référence de ressource inférieurs à 15000 sont réservés à 4D.

Les numéros de référence par défaut des ressources d'éléments de liste sous Mac OS et Windows sont exprimés par les constantes prédéfinies de 4D suivantes :

Constante	Type	Valeur
Réf icône Macintosh	Entier long	860
Réf icône Windows	Entier long	138

4D fournit les ressources 'cicn' suivantes :

Numéro d'ID	Description
860	Sous-liste contractée à la Macintosh
861	Sous-liste déployée à la Macintosh
138	Sous-liste contractée à la Windows
139	Sous-liste déployée à la Windows

Si vous n'utilisez pas le paramètre icône ou passez 0, les éléments sont affichés avec les icônes par défaut pour l'apparence choisie.

Les ressources d'icône couleur peuvent avoir des tailles différentes. Vous pouvez créer, par exemple, des icônes couleur 16x16 ou 32x32.
Si vous ne passez pas le paramètre hauteurLigne, la hauteur de ligne d'une liste hiérarchique sera déterminée par la police et la taille de police utilisées pour l'objet. Si vous utilisez des icônes couleurs qui sont trop grandes ou trop larges, elles seront déformées et/ou tronquées par les lignes pointillées (si l'apparence est Windows) ainsi que par le texte des éléments environnants.

Ajustez en conséquence les tailles des icônes couleurs, les polices et les tailles de police. Vous pouvez également passer dans le paramètre hauteurLigne la hauteur minimale des lignes de la liste hiérarchique. Si la valeur que vous passez est supérieure à la hauteur des lignes définie par la police et la taille de police, elle sera utilisée pour fixer la hauteur des lignes.

Note : CHANGER PROPRIETES LISTE affecte l'apparence de la liste, c'est-à-dire les symboles de déploiement/contraction des éléments et les liens entre les éléments. Si vous voulez personnaliser l'icône de chaque élément d'une liste hiérarchique, utilisez la commande CHANGER PROPRIETES ELEMENT.

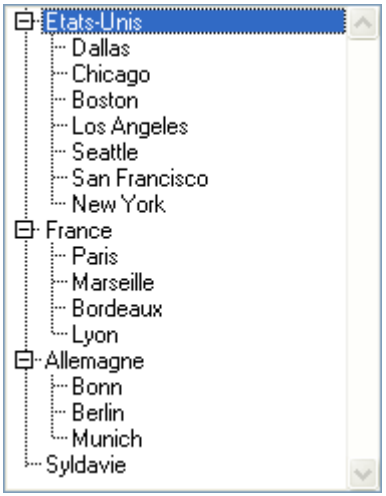
Le paramètre facultatif doubleClic permet d'empêcher que le double-clic sur un élément de la liste ne provoque le déploiement ou la contraction de sa sous-liste.
Par défaut, une sous-liste est déployée ou contractée en cas de double-clic sur l'élément parent. Certains types d'interfaces peuvent toutefois nécessiter une désactivation de ce mécanisme. Pour cela, passez 1 dans le paramètre doubleClic. A noter que seul le double-clic sera désactivé. Les sous-listes pourront toujours être déployées ou contractées par un clic sur l'icône de déploiement.
Si vous passez 0 ou omettez ce paramètre, le fonctionnement par défaut est appliqué.

Le paramètre facultatif multiSélection permet d'indiquer si la liste doit accepter les sélections multiples.
Par défaut, il n'est pas possible de sélectionner simultanément plusieurs éléments d'une liste hiérarchique. Si vous souhaitez que cette fonction soit disponible pour la liste, passez la valeur 1 dans le paramètre multiSélection. Dans ce cas, les sélections multiples peuvent être effectuées :
- manuellement, à l'aide des combinaisons de touches **Maj+clic** pour une sélection continue ou **Ctrl+clic** (Windows) / **Commande+clic** (Mac OS) pour une sélection discontinue,
- par programmation, à l'aide des commandes SELECTIONNER ELEMENTS PAR POSITION et SELECTIONNER ELEMENTS PAR REFERENCE.
Si vous passez 0 ou omettez le paramètre multiSélection, le fonctionnement par défaut est appliqué.

Le paramètre facultatif modifiable permet d'indiquer si la liste sera modifiable par l'utilisateur lorsqu'elle sera affichée sous forme d'énumération associée à un champ ou une variable en saisie. Lorsque l'énumération est modifiable, un bouton **Modifier** est inséré dans la fenêtre d'énumération et l'utilisateur peut ajouter, supprimer et trier les valeurs via un éditeur spécifique.
Si vous passez 1 ou omettez le paramètre modifiable, l'énumération sera modifiable par l'utilisateur ; si vous passez 0, elle ne sera pas modifiable.

Exemples

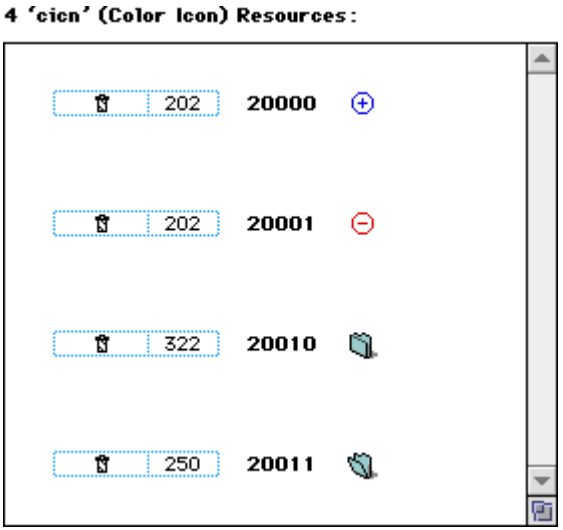
La liste hiérarchique suivante a été définie dans l'éditeur d'énumérations en mode Développement :



Dans un formulaire, l'objet liste hiérarchique hlVilles réutilise cette liste avec cette méthode objet :

```
Au cas ou
    : (Evenement formulaire=Sur chargement)
        hlVilles:=Charger liste("Villes")
        CHANGER PROPRIETES LISTE(hlVilles;vlApparence;vlIcon)
    : (Evenement formulaire=Sur libération)
        SUPPRIMER LISTE(hlVilles;*)
Fin de cas
```

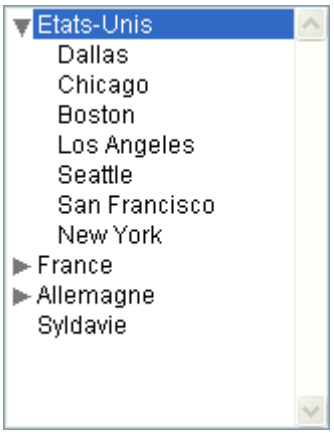
De plus, le fichier de structure de la base a été modifié afin de contenir les ressources d'icônes 'cicn' suivantes :



(1) Avec la ligne suivante :

CHANGER PROPRIETES LISTE(hlVilles;A la Macintosh;Réf icône Macintosh)

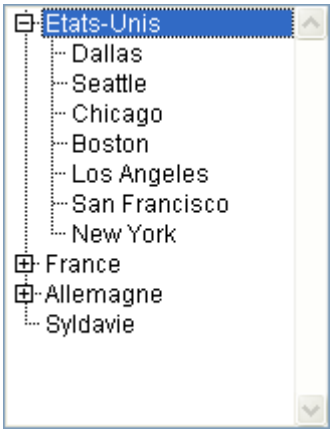
La liste hiérarchique s'affiche ainsi :



(2) Avec la ligne suivante :

CHANGER PROPRIETES LISTE(hlVilles;A la Windows;Réf icône Windows)

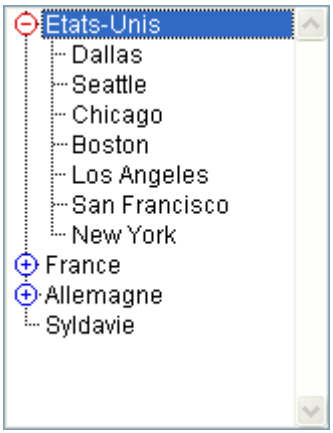
La liste hiérarchique s'affiche ainsi :



(3) Avec la ligne suivante :

CHANGER PROPRIETES LISTE(hlVilles;A la Windows;20000)

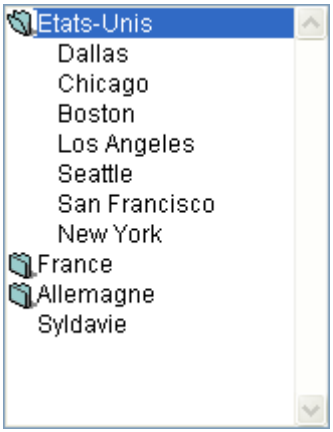
La liste hiérarchique s'affiche ainsi :



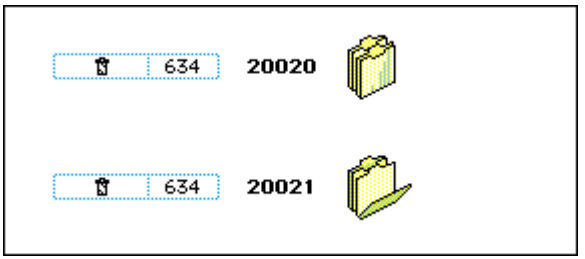
(4) Avec la ligne suivante :

CHANGER PROPRIETES LISTE(hlVilles;A la Macintosh;20010)

La liste hiérarchique s'affiche ainsi :



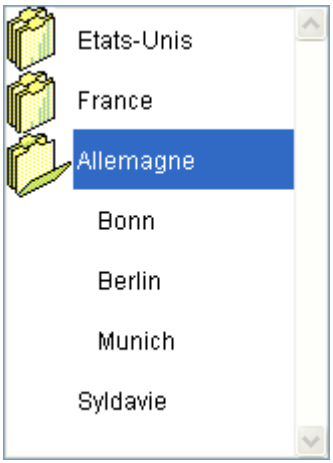
Les ressources d'icône couleur 'cicn' présentées ci-dessous sont ensuite ajoutées au fichier de structure de la base :



(5) Avec la ligne suivante :

CHANGER PROPRIETES LISTE(hlVilles;A la Windows;20020;32)

La liste hiérarchique s'affiche ainsi :



Référence

CHANGER PROPRIETES ELEMENT, LIRE PROPRIETES ELEMENT, LIRE PROPRIETES LISTE.

Charger liste (nomListe) → RéfListe

Paramètre	Type		Description
nomListe	Alpha	→	Nom de liste créée dans l'éditeur d'énumérations
Résultat	RéfListe	←	Numéro de référence de la liste nouvellement créée

Description

La commande Charger liste crée une liste hiérarchique dont le contenu est copié depuis la liste nomListe créée en mode Développement, dans l'éditeur d'énumérations. La fonction retourne le numéro de référence de la liste nouvellement créée.

Pour connaître les énumérations définies dans la base, utilisez la commande LISTE ENUMERATIONS. Pour savoir si la liste a correctement été chargée, utilisez la fonction Liste existante avec le numéro de référence retourné par Charger liste.

Notez que la nouvelle liste est une copie de la liste définie en mode Développement. Par conséquent, toute modification apportée à cette nouvelle liste n'affectera pas la liste définie en mode Développement. De même, toute modification ultérieure de l'énumération n'affecte pas la liste que vous venez de créer.

Si vous modifiez la liste nouvellement créée et voulez enregistrer ces modifications, utilisez la commande STOCKER LISTE.

Si vous n'avez plus besoin de la liste, n'oubliez pas d'appeler SUPPRIMER LISTE pour la supprimer. Sinon, elle reste en mémoire jusqu'à la fin de la session de travail ou jusqu'à ce que le process dans lequel la liste a été créée soit détruit.

Astuce : Si vous associez une liste à un objet de formulaire (liste hiérarchique, onglet ou menu hiérarchique) à l'aide du menu **Enumération** dans la Liste des propriétés, il est inutile d'appeler Charger liste ou SUPPRIMER LISTE dans la méthode de l'objet. 4D charge et efface la liste automatiquement pour vous.

Exemple

Imaginons que vous créez une base pour le marché international. Vous voulez pouvoir changer la langue utilisée. Dans un formulaire, vous présentez une liste hiérarchique listeHL qui propose les langues disponibles. En mode Développement, vous avez préparé des listes différentes, par exemple “Options US” pour la version anglaise, “Options FR” pour la version française, “Options ES” pour la version espagnole, etc. De plus, vous maintenez la variable interprocess <>gaLangueCourante dans laquelle vous stockez un code de langue sur 2 caractères, par exemple “US” pour la version anglaise, “FR” pour la version française, “ES” pour la version espagnole, etc. Pour vous assurer que la liste correcte sera chargée en utilisant la langue choisie, vous pouvez écrire :

```
    ` Méthode objet de la liste hiérarchique listeHL
Au cas ou
    : (Evenement formulaire = Sur chargement)
        C_ENTIER LONG (listeHL)
        listeHL:=Charger liste("Options"+<>gaLangueCourante)
    : (Evenement formulaire = Sur libération)
        SUPPRIMER LISTE(listeHL;*)
Fin de cas
```

Référence

Liste existante, STOCKER LISTE, SUPPRIMER LISTE.

Chercher dans liste ({*; }liste; valeur; portée{; tabEléments{; *}}) → Entier long

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
valeur	Chaîne	→	Valeur à rechercher
portée	Entier	→	0=Liste principale, 1=Sous-listes
tabEléments	Tab Entier long	←	- Si 2e * omis : tableau des positions des éléments trouvés - Si 2e * passé : tableau des numéros de référence des éléments trouvés
*	*	→	- Si omis : utiliser la position des éléments - Si passé : utiliser le numéro de référence des éléments
Résultat	Entier long	←	- Si 2e * omis : position de l'élément trouvé - Si 2e * passé : numéro de référence de l'élément trouvé

Description

La commande Chercher dans liste retourne la position ou la référence du premier élément de liste qui équivaut à la chaîne passée dans valeur. Si plusieurs éléments sont trouvés, la fonction peut également remplir le tableau tabEléments avec la position ou la référence de chaque élément.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les numéros de référence des éléments (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec les positions des éléments (le second * est passé), la syntaxe basée sur le nom d'objet est requise car la position des éléments peut varier d'une représentation à l'autre.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande Chercher dans liste s'appliquera au premier objet dont le nom correspond.

Le second paramètre * vous permet d'indiquer si vous souhaitez travailler avec des positions courantes d'éléments (dans ce cas, ce paramètre doit être omis) ou des références absolues d'éléments (dans ce cas, il doit être passé).

Passez dans valeur la chaîne de caractères à rechercher. La recherche sera du type "est exactement", c'est-à-dire que la recherche de "bois"ne trouvera pas "boissons". Toutefois, vous pouvez utiliser le caractère @ pour définir des recherches du type "commence par", "se termine par" ou "contient".

Le paramètre portée vous permet de définir si la recherche doit porter uniquement sur le premier niveau de la liste ou si elle doit inclure toutes ses sous-listes. Passez 0 pour concentrer la recherche sur le premier niveau de la liste et 1 pour l'étendre à toutes les sous-listes.

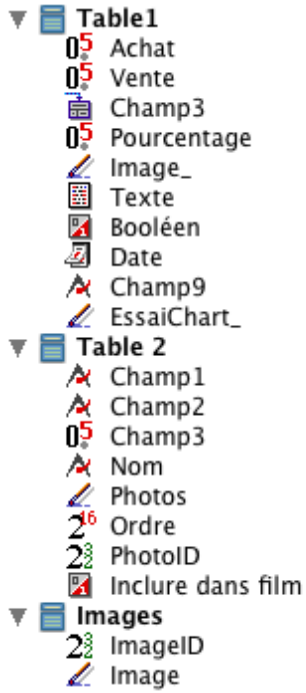
Si vous souhaitez connaître la position ou le numéro de tous les éléments correspondant à valeur, passez un tableau d'entiers longs dans le paramètre facultatif tabEléments. Si nécessaire, le tableau sera créé et redimensionné par la commande. La commande remplira le tableau avec les positions (si le second * est omis) ou les numéros de référence (si le second * est passé) des éléments trouvés.

Les positions sont exprimées relativement à l'élément supérieur de la liste principale, en tenant compte de l'état courant déployé/contracté de la liste et de ses sous-listes.

Si aucun élément ne correspond à la valeur recherchée, la fonction retourne 0 et le tableau tabEléments est retourné vide.

Exemple

Soit la liste hiérarchique suivante :




```
$vlItemPos:=Chercher dans liste(hList;"P@";1; $arrPos)
    `$vlItemPos vaut 5
    `$arrPos{1} vaut 5, $arrPos{2} vaut 17 et $arrPos{3} vaut 19
$vlItemRef:=Chercher dans liste(hList;"P@";1;$arrRefs;*)
    `$vlItemRef vaut 7
    `$arrRefs{1} vaut 7, $arrRefs{2} vaut 18 et $arrRefs{3} vaut 23
$vlItemPos:=Chercher dans liste(hList;"Date";1;$arrPos)
    `$vlItemPos vaut 9
    `$arrPos{1} vaut 9
$vlItemRef:Chercher dans liste(hList;"Date";1;$arrRefs;*)
    `$vlItemRef vaut 11
    `$arrRefs{1} vaut 11
$vlItemPos:=Chercher dans liste(hList;"Date";0;*)
    `$vlItemPos vaut 0  `
```

Copier liste

Copier liste (liste) → RéfListe

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de la liste à copier
Résultat	RéfListe	←	Numéro de référence de la nouvelle liste

Description

La commande Copier liste duplique la liste dont vous passez le numéro de référence dans le paramètre liste et retourne le numéro de référence de la nouvelle liste.

Le contenu de la liste copiée est entièrement dupliqué. Une fois que vous en avez terminé avec la copie de la liste, appelez la commande SUPPRIMER LISTE pour l'effacer.

Référence

Charger liste, Nouvelle liste, SUPPRIMER LISTE.

Element parent ({*; }liste; réfElément | *) → Entier long

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
Résultat	Entier long	←	Numéro de référence de l'élément parent ou 0 s'il n'y en a pas

Description

La commande Element parent retourne le numéro de référence de l'élément parent.

Passez dans liste le numéro de référence ou le nom d'objet de la liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande Element parent s'appliquera au premier objet dont le nom correspond.

Passez dans réfElément le numéro de référence d'un élément de la liste ou 0, ou encore *. Si vous passez 0, la commande s'applique au dernier élément ajouté à la liste. Si vous passez *, la commande s'applique à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés, l'élément courant est celui qui a été sélectionné en dernier.

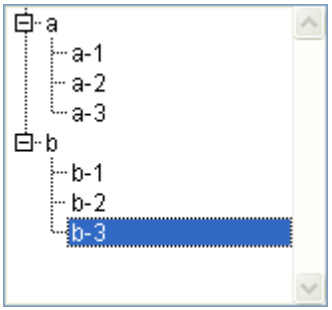
En retour, si un élément correspondant existe bien dans la liste et si cet élément se trouve bien dans une sous-liste (et a donc un élément parent), vous récupérez le numéro de référence de l'élément parent.

S'il n'existe pas d'élément numéro réfElément, ou si vous avez passé * et qu'aucun élément n'est sélectionné, ou si cet élément n'a pas d'élément parent, Element parent retourne 0 (zéro).

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la description de la commande AJOUTER A LISTE.

Exemples

Voici une liste hList affichée en mode Application :



Voici les numéros de référence des éléments de cette liste :

Elément	Numéro
a	100
a-1	101
a-2	102
b	200
b-1	201
b-2	202
b-3	203

• Avec le code ci-dessous, si l'élément “b-3” est sélectionné, la variable $vlParentElémRef prend la valeur 200, c'est-à-dire le numéro de référence de l'élément “b” :

```
$vlElémPos:=Elements selectionnes(hList)
INFORMATION ELEMENT(hList;$vlElémPos;$vlElémRef;$vsItemText)
$vlParentElémRef:=Element parent(hList;$vlElémRef) ` $vlParentElémRef vaut 200
```

• Si l'élément “a-1” était sélectionné, la variable $vlParentElémRef prendrait la valeur 100, c'est-à-dire le numéro de référence de l'élément “a”.

• Si l'élément “a” ou “b” était sélectionné, la variable $vlParentElémRef prendrait la valeur 0 car ces éléments n'ont pas d'élément parent.

Référence

CHANGER ELEMENT, INFORMATION ELEMENT, Position element liste, SELECTIONNER ELEMENTS PAR REFERENCE.

Elements selectionnes ({*; }liste{; tabEléments{; *}}) → Entier long

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
tabEléments	Tab Entier long	←	Si 2e * omis : Tableau des positions des éléments sélectionnés dans la ou les liste(s) Si 2e * passé : Tableau des références des éléments sélectionnés dans la ou les liste(s)
*	*	→	Si omis : Position(s) d'élément(s) Si passé : Référence(s) d'élément(s)
Résultat	Entier long	←	Si 2e * omis : Position de l'élément sélectionné parmi la ou les liste(s) déployée(s)/contractée(s) Si 2e * passé : Référence de l'élément sélectionné

Description

La fonction Elements selectionnes retourne la **position** ou la **référence** de l'élément sélectionné dans la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les références d'éléments (le second * est passé), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec les positions des éléments (le second * est omis), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration d'éléments déployés/contractés.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande Elements selectionnes s'appliquera au premier objet dont le nom correspond.

En cas de sélection multiple, la fonction peut également retourner dans le tableau tabEléments la position ou la référence de chaque élément sélectionné. Cette fonction doit être appliquée à une liste affichée dans un formulaire afin de détecter le ou les élément(s) sélectionné(s) par l'utilisateur.

Le second paramètre * permet d'indiquer si vous souhaitez travailler avec des positions courantes d'éléments (dans ce cas, ce paramètre doit être omis) ou des références absolues d'éléments (dans ce cas, il doit être passé).

Vous pouvez passer dans le paramètre tabEléments un tableau d'entiers longs. Si nécessaire, le tableau sera créé et redimensionné par la commande. A l'issue de l'exécution de la commande, tabEléments contiendra :
• la position de chaque élément sélectionné relativement à l'état déployé/contracté de la ou des liste(s) si le paramètre * est omis.
• la référence absolue de chaque élément sélectionné si le paramètre * est passé.
Le tableau est retourné vide si aucun élément n'est sélectionné.

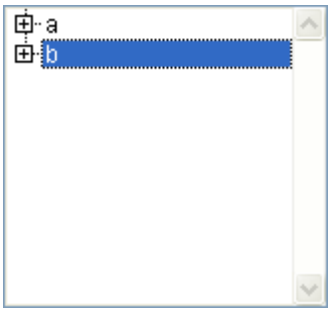
Note : En cas de sélection multiple, la commande retourne la position ou la référence de l'élément courant de liste. L'élément courant est le dernier élément sur lequel l'utilisateur a cliqué (sélection manuelle) ou l'élément désigné par la commande SELECTIONNER ELEMENTS PAR POSITION ou SELECTIONNER ELEMENTS PAR REFERENCE (sélection par programmation).

Si la liste comporte des sous-listes, appliquez la fonction à la liste principale (celle qui est associée au formulaire), et non à une de ses sous-listes. Les positions sont exprimées relativement à l'élément supérieur de la liste principale, en tenant compte de l'état courant déployé/contracté de la liste et de ses sous-listes.

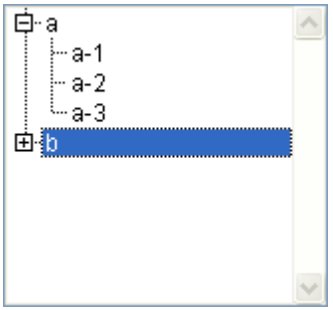
Dans tous les cas, si aucun élément n'est sélectionné, la fonction retourne 0.

Exemple

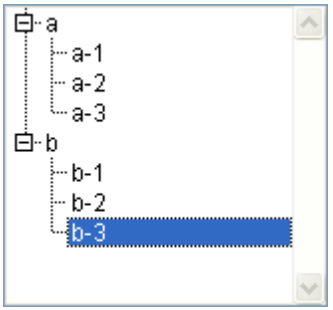
Voici la liste hList telle qu'elle apparaît en mode Application :



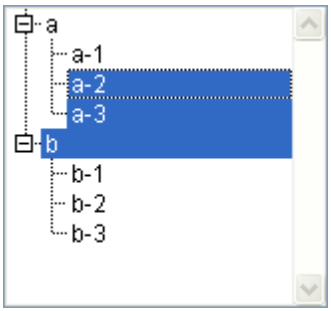
$vlItemPos:=**Elements selectionnes**(hList) ` à ce stade, $vlItemPos vaut 2



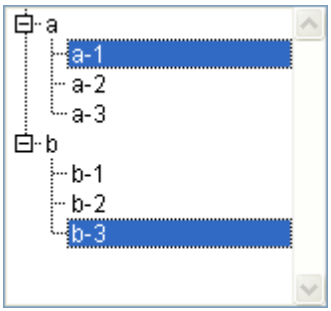
$vlItemPos:=**Elements selectionnes**(hList) ` à ce stade, $vlItemPos vaut 5
$vlItemRef:=**Elements selectionnes**(hList;*) ` $vlItemRef vaut 200 (par exemple)



$vlItemPos:=**Elements selectionnes**(hList) ` à ce stade, $vlItemPos vaut 8
$vlItemRef:=**Elements selectionnes**(hList;*) ` $vlItemRef vaut 203 (par exemple)



$vlItemPos:=**Elements selectionnes**(hList;$tabPos) ` à ce stade, $vlItemPos vaut 3
` $tabPos{1} vaut 3, $tabPos{2} vaut 4 et $tabPos{3} vaut 5



$vlItemRef:=**Elements selectionnes**(hList;$tabRefs;*) ` $vlItemRef vaut 203 (par exemple)
` $tabRefs{1} vaut 101, $tabRefs{2} vaut 203 (par exemple)

Référence

SELECTIONNER ELEMENTS PAR POSITION, SELECTIONNER ELEMENTS PAR REFERENCE.

FIXER ICONE ELEMENT ({*; }liste; refElément | *; icône)

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
refElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
icône	Image	→	Icône à associer à l'élément

Description

La commande FIXER ICONE ELEMENT permet de modifier l'icône associée à l'élément désigné par le paramètre réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Note : Il est possible de modifier l'icône associée à un élément à l'aide de la commande CHANGER PROPRIETES ELEMENT. Toutefois, CHANGER PROPRIETES ELEMENT accepte uniquement des références d'images statiques (références de ressources ou images de la bibliothèque).

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.Vous pouvez également passer 0 dans réfElément afin de demander la modification du dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE).
Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Passez dans le paramètre icône une expression image 4D valide (champ, variable, pointeur, etc.). L'image sera placée à gauche de l'élément.
L'emploi de pointeurs est particulièrement recommandé car les listes hiérarchiques sont optimisées dans ce cas : une seule instance de l'image sera créée en mémoire si la même icône est utilisée pour plusieurs éléments de la liste.

Note : A l'inverse, l'emploi direct de variables générées par les commandes LIRE RESSOURCE ICONE ou LIRE RESSOURCE IMAGE est déconseillé car l'icône sera dupliquée en mémoire pour chaque élément de la liste.

Exemple

Ce code est optimisé grâce à l'emploi d'un pointeur :

```
vIcon:=->[Params]Icone
FIXER ICONE ELEMENT(maliste;ref1;vIcon->)
FIXER ICONE ELEMENT(maliste;ref2;vIcon->)
```

Référence

CHANGER PROPRIETES ELEMENT.

FIXER PARAMETRE ELEMENT ({*; }liste; réfElément | *; sélecteur; valeur)

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
sélecteur	Chaîne	→	Constante de paramètre
valeur	Chaîne|Booléen|Num	→	Valeur de paramètre

Description

La commande FIXER PARAMETRE ELEMENT permet de modifier le paramètre sélecteur pour l'élément réfElément de la liste hiérarchique dont vous avez passé la référence ou le nom d'objet dans le paramètre liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.Vous pouvez également passer 0 dans réfElément afin de demander la modification du dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE).
Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Vous pouvez passer dans sélecteur la constante Texte supplémentaire (placée dans le thème "Listes hiérarchiques") ou toute valeur personnalisée :
• Texte supplémentaire : cette constante permet d'ajouter un texte à droite de l'élément réfElément. Ce libellé supplémentaire reste toujours affiché dans la partie droite de la liste, même si l'utilisateur déplace le curseur de défilement horizontal. Lorsque vous utilisez cette constante, passez dans valeur le texte à afficher.
• Sélecteur personnalisé : vous pouvez passer dans sélecteur tout texte personnalisé et lui associer une valeur de type texte, numérique ou booléen. Cette valeur sera stockée avec l'élément et pourra être récupérée via la commande LIRE PARAMETRE ELEMENT. Ce principe permet de mettre en place tout type d'interface associée aux listes hiérarchiques. Par exemple, dans une liste stockant des noms de personnes, vous pouvez stocker l'âge de chaque personne et ne l'afficher que lorsque l'élément correspondant est sélectionné.

Référence

LIRE PARAMETRE ELEMENT.

FIXER POLICE ELEMENT ({*; }liste; réfElément | *; police)

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfList | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
police	Chaîne | Num	→	Nom ou numéro de police

Description

La commande FIXER POLICE ELEMENT modifie la police de caractères de l'élément désigné par le paramètre réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien.Vous pouvez également passer 0 dans réfElément afin de demander la modification du dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE).
Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Passez dans le paramètre police le nom ou le numéro de la police à utiliser. Pour réappliquer la police par défaut de la liste hiérarchique, passez une chaîne vide dans police.

Exemple

Appliquer la police Times à l'élément courant de la liste :

```
FIXER POLICE ELEMENT(*;"Maliste";*;"Times")
```

Référence

CHANGER JEU DE CARACTERES, Lire police element.

INFORMATION ELEMENT ({*; }liste; positionElém | *; réfElément; textElém{; sous_Liste{; déployé}})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
positionElém | *	Numérique | *	→	Position de l'élément dans la ou les liste(s) déployée(s)/contractée(s) ou * pour l'élément courant de la liste
réfElément	Entier long	←	Numéro de référence de l'élément
textElém	Alpha	←	Libellé de l'élément
sous_Liste	RéfListe	←	Numéro de référence de sous-liste (s'il y en a)
déployé	Booléen	←	Si une sous-liste est rattachée à l'élément : Vrai = la sous-liste est déployée Faux = la sous-liste est contractée

Description

La commande INFORMATION ELEMENT retourne des informations sur l'élément désigné par le paramètre positionElém de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste, vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste, la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée et de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande INFORMATION ELEMENT s'appliquera au premier objet dont le nom correspond.

La position doit être exprimée relativement à l'état déployé/contracté de la liste et de ses sous-listes. Vous devez passer une valeur de position comprise entre 1 et la valeur retournée par Nombre elements. Si vous passez une valeur située hors de cet intervalle, INFORMATION ELEMENT retourne des valeurs vides (0, "", etc.).
Si vous passez * dans positionElém, la commande s'applique à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande retourne des valeurs vides.

Après l'appel, vous récupérez :
• Le numéro de référence de l'élément dans réfElément.
• Le libellé de l'élément dans textElém.

Si vous passez les paramètres optionnels sous_Liste et déployé :
• sous_Liste contient le numéro de référence de la sous-liste rattachée à l'élément. Si l'élément n'a pas de sous-liste associée, sous_Liste retourne zéro.
• Si l'élément comporte une sous-liste, déployé retourne Vrai si la sous-liste est déployée, et Faux sinon.

Exemples

(1) En partant de l'hypothèse que hList est une liste dont les éléments ont des numéros de référence uniques, le code suivant inverse automatiquement l'état déployé/contracté de la sous-liste, si elle existe, rattachée à l'élément sélectionné :

```
C_BOOLEEN($vbDéployé)
C_ENTIER LONG($hSousListe;$vlElemRef)
C_ALPHA(31;$vsElemText)
    `La déclaration de ces variables est nécessaire si vous souhaitez compiler la méthode
$vlElemPos:=Elements selectionnes(hList)
Si ($vlElemPos>0)
    INFORMATION ELEMENT(hList;$vlElemPos;$vlElemRef;$vsElemText;$hSousListe;
                                                                $vbDéployé)
    Si (Liste existante($hSousListe))
        CHANGER ELEMENT(hList;$vlElemRef;$vsElemText;$hSousListe;Non($vbDéployé))
    Fin de si
Fin de si
```

(2) Reportez-vous à l'exemple de la commande AJOUTER A LISTE.

Référence

CHANGER ELEMENT, CHANGER PROPRIETES ELEMENT, Element parent, Elements selectionnes, LIRE PROPRIETES ELEMENT, Position element liste.

INSERER DANS LISTE({*; } liste; avantElément | *; texteElément; réfElément{; sous_Liste; déployé})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
avantElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément de la liste actuellement sélectionné
texteElément	Chaîne	→	Libellé du nouvel élément
réfElément	Entier long	→	Numéro de référence unique du nouvel élément
sous_Liste	RefList	→	Sous-liste optionnelle rattachée au nouvel élément
déployé	Booléen	→	Indique si la sous-liste doit être déployée ou non

Description

La commande INSERER DANS LISTE insère l'élément désigné par le paramètre réfElément dans la liste dont le numéro de référence ou le nom d'objet est passé dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Le paramètre avantElément permet de désigner l'élément avant lequel vous souhaitez insérer le nouvel élément :
• Vous pouvez passer la valeur 0 afin de désigner le dernier élément ajouté à la liste. Le nouvel élément devient l'élément sélectionné.
• Vous pouvez passer * afin que le nouvel élément soit inséré avant l'élément actuellement sélectionné dans la liste. Le nouvel élément devient l'élément sélectionné.

• Si vous souhaitez insérer le nouvel élément avant un élément spécifique, passez le numéro de référence de cet élément comme deuxième paramètre. Dans ce cas, le nouvel élément inséré n'est pas automatiquement sélectionné. Si le numéro que vous passez ne correspond à aucun élément de la liste, la commande ne fait rien.

Vous passez le texte et le numéro de référence du nouvel élément dans les paramètres texteElément et réfElément.

Si vous souhaitez que l'élément comporte des sous-éléments, passez un numéro de référence de liste valide dans le paramètre sous_Liste. Dans ce cas, vous devez également passer le paramètre déployé. Passez Vrai ou Faux dans ce paramètre pour que cette sous-liste s'affiche respectivement déployée ou contractée.

Exemple

L'exemple suivant insère un élément (associé à aucune sous-liste) juste devant l'élément actuellement sélectionné dans la liste hList:

```
vlUniqueRef:=vlUniqueRef+1
INSERER DANS LISTE(hList;*;"Nouvel élément";vlUniqueRef)
```

Référence

AJOUTER A LISTE.

LIRE ICONE ELEMENT ({*; }liste; réfElément | *; icône)

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
icône	Var image	←	Icône associée à l'élément

Description

La commande LIRE ICONE ELEMENT retourne dans icône l'icône associée à l'élément dont vous avez passé le numéro de référence dans réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande LIRE ICONE ELEMENT s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans réfElément afin de désigner le dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE). Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Passez dans icône une variable image. A l'issue de l'exécution de la commande, elle contiendra l'icône associée à l'élément, quelle que soit la source de l'icône (image statique, ressource ou expression image).

Si aucune icône n’est associée à l’élément, la variable icône est retournée vide.

Note : Lorsque l'icône associée à un élément a été définie via une référence statique (références de ressources ou images de la bibliothèque), il est possible de connaître son numéro à l’aide de la commande LIRE PROPRIETES ELEMENT.

Référence

FIXER ICONE ELEMENT, LIRE PROPRIETES ELEMENT.

LIRE PARAMETRE ELEMENT ({*; }liste; réfElément | *; sélecteur; valeur)

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
sélecteur	Chaîne	→	Constante de paramètre
valeur	Chaîne|Booléen|Num	←	Valeur courante du paramètre

Description

La commande LIRE PARAMETRE ELEMENT permet de connaître la valeur courante du paramètre sélecteur pour l'élément réfElément de la liste hiérarchique dont vous avez passé la référence ou le nom d'objet dans le paramètre liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande LIRE PARAMETRE ELEMENT s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans réfElément afin de désigner le dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE). Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Vous pouvez passer dans sélecteur la constante Texte supplémentaire (placée dans le thème “Listes hiérarchiques”) ou toute valeur personnalisée. Pour plus d’informations sur les paramètres sélecteur et valeur, reportez-vous à la description de la commande FIXER PARAMETRE ELEMENT.

Référence

FIXER PARAMETRE ELEMENT.

Lire police element ({*; }liste; réfElément | *) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est un numéro de référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
Résultat	Chaîne	←	Nom de police

Description

La commande Lire police element retourne le nom de la police de caractères courante de l'élément désigné par le paramètre réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande Lire police element s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer un numéro de référence dans réfElément. Si ce numéro ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer 0 dans réfElément afin de désigner le dernier élément ajouté à la liste (à l'aide de AJOUTER A LISTE). Vous pouvez enfin passer * dans réfElément : dans ce cas, la commande s'appliquera à l'élément courant de la liste. Si plusieurs éléments sont sélectionnés manuellement, l'élément courant est celui qui a été sélectionné en dernier. Si aucun élément n'est sélectionné, la commande ne fait rien.

Référence

FIXER POLICE ELEMENT.

LIRE PROPRIETES ELEMENT ({*; }liste; réfElément | *; saisissable{; styles{; icône{; couleur}}})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément courant de la liste
saisissable	Booléen	←	Vrai = Saisissable, Faux = Non-saisissable
styles	Numérique	←	Style de police de l'élément
icône	Numérique	←	Numéro de ressource Mac OS 'cicn' ou 65536 + numéro de ressource Mac OS 'PICT' ou 131072 + numéro de référence d'image
couleur	Entier long	←	Valeur de couleur RVB

Description

La commande LIRE PROPRIETES ELEMENT retourne les propriétés de l'élément désigné par le paramètre réfElément de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande LIRE PROPRIETES ELEMENT s'appliquera au premier objet dont le nom correspond.

Vous pouvez passer dans réfElément un numéro de référence, la valeur 0 afin de désigner le dernier élément ajouté à la liste ou encore * afin de désigner l'élément courant de la liste. Si plusieurs éléments sont sélectionnés, l'élément courant est celui qui a été sélectionné en dernier.
Si vous passez * et qu'aucun élément n'est sélectionné ou si le numéro de référence d'élément ne correspond à aucun élément de la liste, la commande laisse les paramètres inchangés.

Si vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la description de la commande AJOUTER A LISTE.

Après l'appel :
• saisissable retourne Vrai si l'élément est saisissable.
• styles retourne le style de caractères de l'élément.
• icône retourne l'icône ou l'image associée à l'élément, et 0 s'il n'y en a pas.
• couleur retourne la couleur du texte de l'élément désigné.

Note : Vous pouvez récupérer dans une variable image l'icône associée à un élément à l'aide de la commande LIRE ICONE ELEMENT.

Pour plus d'informations sur ces propriétés, reportez-vous à la description de la commande CHANGER PROPRIETES ELEMENT.

Référence

CHANGER ELEMENT, CHANGER PROPRIETES ELEMENT, INFORMATION ELEMENT, LIRE ICONE ELEMENT.

LIRE PROPRIETES LISTE (liste; apparence{; icône{; hauteurLigne{; doubleClic{; multiSélection{; modifiable}}}}})

Paramètre	Type		Description
liste	RefListe	→	Numéro de référence de la liste
apparence	Numérique	←	Style graphique de la liste 1 = Liste hiérarchique à la Macintosh 2 = Liste hiérarchique à la Windows
icône	Numérique	←	Référence de ressource Mac OS ‘cicn’
hauteurLigne	Numérique	←	Hauteur minimale de la ligne (pixels)
doubleClic	Entier long	←	Déploiement/contraction sur double-clic 0 = autorisé, 1= empêché
multiSélection	Entier long	←	Sélections multiples : 0 = interdites, 1 = autorisées
modifiable	Entier long	←	Enumération modifiable : 0 = non, 1 = oui

Description

La commande LIRE PROPRIETES LISTE retourne des informations sur la liste hiérarchique dont vous avez passé le numéro de référence dans le paramètre liste.

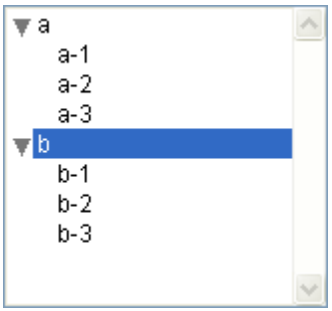
Le paramètre apparence retourne le style graphique de la liste.
Le paramètre icône retourne les icônes utilisées pour symboliser l'état déployé/contracté d'une sous-liste.
Le paramètre hauteurLigne retourne la hauteur de ligne minimale.
Si le paramètre doubleClic vaut 1, le déploiement ou la contraction des sous-listes en cas de double-clic sur l’élément parent est désactivé(e). Si doubleClic vaut 0, ce fonctionnement est actif (valeur par défaut).
Si le paramètre multiSélection vaut 0, la sélection multiple d’éléments (manuelle ou par programmation) n’est pas possible dans la liste. S’il vaut 1, la sélection multiple est permise.
Si le paramètre modifiable vaut 1, la liste est modifiable lorsqu’elle est affichée sous forme d’énumération dans les enregistrements. S’il vaut 0, la liste n’est pas modifiable.

Ces propriétés peuvent être définies à l'aide de la commande CHANGER PROPRIETES LISTE et/ou dans l'éditeur d'énumérations en mode Développement, si la liste a été créée dans cet éditeur ou sauvegardée avec la commande STOCKER LISTE.

Pour une description complète de ces propriétés d'apparence et de comportement, reportez-vous à la commande CHANGER PROPRIETES LISTE.

Exemple

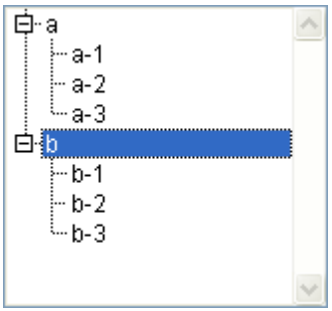
Voici une liste nommée hList, telle qu'elle apparaît en mode Application :



Voici la méthode objet d'un bouton :

```
    ` Méthode objet du bouton bMacOuWin
LIRE PROPRIETES LISTE(hList;$vlApparence;$vlIcon;$vlLH;$vlClic;$vlSelect;$vlModif)
Si ($vlApparence=A la Macintosh)
    $vlApparence:=A la Windows
    $vlIcon:=Réf icône Windows
    $vlModif:=1
Sinon
    $vlApparence:=A la Macintosh
    $vlIcon:=Réf icône Macintosh
    $vlModif:=1
Fin de si
CHANGER PROPRIETES LISTE(hList;$vlApparence;$vlIcon;$vlLH;$vlClic;$vlSelect;$vlModif)
```

Cette méthode permet d'afficher la liste ainsi :



Référence

CHANGER PROPRIETES LISTE.

LISTE ENUMERATIONS (tabNums; tabNoms)

Paramètre	Type		Description
tabNums	Tab Entier long	←	Numéros des énumérations
tabNoms	Tab Texte	←	Noms des énumérations

Description

La commande LISTE ENUMERATIONS retourne dans les tableaux synchronisés tabNums et tabNoms les numéros et les noms des énumérations définies dans l'éditeur d'énumérations en mode Développement.

Les numéros des énumérations correspondent à leur ordre de création. Dans l'éditeur d'énumérations, les énumérations sont affichées par ordre alphabétique.

Liste existante (liste) → Booléen

Paramètre	Type		Description
liste	RéfListe	→	Référence de la liste à tester
Résultat	Booléen	←	Vrai si liste est une liste hiérarchique Faux si liste n'est pas une liste hiérarchique

Description

La fonction Liste existante retourne VRAI si la valeur passée dans le paramètre liste est une référence valide à une liste hiérarchique. Dans les autres cas, elle retourne FAUX.

Exemples

(1) Reportez-vous à l'exemple de la commande SUPPRIMER LISTE.
(2) Reportez-vous aux exemples de la commande PROPRIETES GLISSER DEPOSER.

Référence

PROPRIETES GLISSER DEPOSER.

Nombre elements ({*; }liste{; *}) → Entier long

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
*	*	→	Si omis (défaut) : Retourner les éléments visibles (déployés) dans la ou les liste(s) Si spécifié : Retourner tous les éléments
Résultat	Entier long	←	Nombre d'éléments visibles (déployés) si 2e * omis ou Nombre total d'éléments si 2e * passé

Description

La fonction Nombre elements retourne soit le nombre d'éléments visibles soit le nombre total d'éléments dans la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec tous les éléments (le second * est passé), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec les éléments visibles (le second * est omis), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande Nombre elements s'appliquera au premier objet dont le nom correspond.

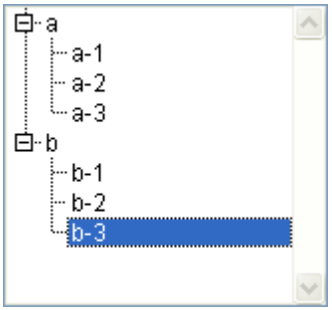
Le choix du type d'information à retourner est effectué à l'aide du second paramètre *. Lorsque ce paramètre est passé, la commande retourne le nombre total d'éléments présents dans la liste, quel que soit son état courant déployé/contracté.

Lorsque ce paramètre est omis, la commande retourne le nombre d'éléments qui sont visibles, en fonction de l'état déployé/contracté actuel de la liste et de ses sous-listes.

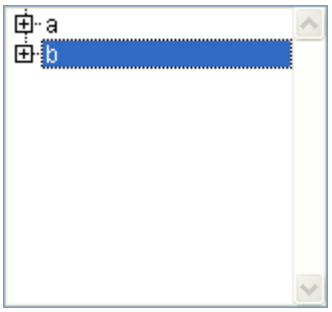
Cette fonction doit être appliquée à une liste affichée dans un formulaire.

Exemples

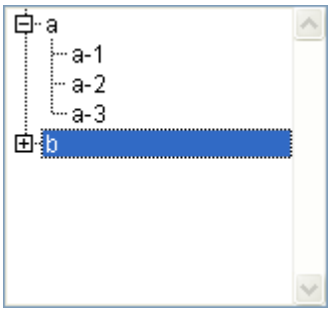
Voici la liste hList affichée en mode Application :




```
$vlNbItems:=Nombre elements(hList)  ` à ce stade, $vlNbItems vaut 8
$vlNbTItems:=Nombre elements(hList;*)  `$vlNbTItems vaut également 8
```




```
$vlNbItems:=Nombre elements(hList)  ` à ce stade, $vlNbItems vaut 2
$vlNbTItems:=Nombre elements(hList;*)  `$vlNbTItems vaut toujours 8
```




```
$vlNbItems:=Nombre elements(hList)  `$vlNbItems vaut 5
$vlNbTItems:=Nombre elements(hList;*)  `$vlNbTItems vaut toujours 8
```

Référence

Elements selectionnes, Position element liste.

Nouvelle liste → RéfListe

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	RéfListe	←	Numéro de référence de liste

Description

La commande Nouvelle liste crée une nouvelle liste hiérarchique vide en mémoire et retourne son numéro de référence unique.

ATTENTION : Les listes hiérarchiques résident en mémoire. Une fois que vous en avez terminé avec une liste hiérarchique, il est important que vous l'effaciez à l'aide de la commande SUPPRIMER LISTE. Ainsi, vous libérez la mémoire occupée par la liste hiérarchique dont vous n'avez plus besoin.

D'autres commandes vous permettent de créer des listes hiérarchiques :
• Copier liste crée une nouvelle liste en dupliquant une liste existante.
• Charger liste crée une nouvelle liste en chargeant une énumération créée (manuellement ou par programmation) dans l'éditeur d'énumérations du mode Développement.
• BLOB vers liste crée une nouvelle liste à partir du contenu d'un BLOB dans lequel une liste avait été préalablement stockée.

Une fois que vous avez créé une liste hiérarchique à l'aide de la commande Nouvelle liste, vous pouvez :
• Ajouter des éléments à la liste à l'aide des commandes AJOUTER A LISTE et INSERER DANS LISTE.
• Supprimer des éléments de cette liste à l'aide de la commande SUPPRIMER DANS LISTE.

Exemple

Reportez-vous à l'exemple de la commande AJOUTER A LISTE.

Référence

AJOUTER A LISTE, BLOB vers liste, Charger liste, Copier liste, INSERER DANS LISTE, LISTE VERS BLOB, SUPPRIMER DANS LISTE, SUPPRIMER LISTE.

Position element liste

Position element liste ({*; }liste; réfElément) → Numérique

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément	Entier long	→	Numéro de référence d'élément
Résultat	Numérique	←	Position de l'élément parmi la ou les liste(s) déployée(s)/contractée(s)

Description

La commande Position element liste retourne la position de l'élément dont vous avez passé le numéro de référence dans réfElément parmi la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste, vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste, la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande Position element liste s'appliquera au premier objet dont le nom correspond.

Note : A la différence des autres commandes de ce thème, cette commande ne permet pas de passer la valeur 0 dans réfElément pour désigner le dernier élément ajouté.

La position est exprimée relativement à l'élément supérieur de la liste, en tenant compte de l'état déployé/contracté de la liste et de ses sous-listes.

Le résultat est donc compris entre 1 et la valeur retournée par Nombre elements.

Si l'élément n'est pas visible car il est inclus dans une liste contractée, Position element liste déploie la liste correspondante de manière à ce que l'élément devienne visible.

Si l'élément n'existe pas, Position element liste retourne 0.

Référence

Nombre elements, SELECTIONNER ELEMENTS PAR REFERENCE.

REDESSINER LISTE (liste)

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de liste

Note de compatibilité : La commande REDESSINER LISTE est inutile à compter de la version 11 de 4D. Toutes les représentations des listes hiérarchiques sont désormais automatiquement redessinées. Lorsqu'elle est appelée, cette commande ne fait rien.

SELECTIONNER ELEMENTS PAR POSITION ({*; }liste; positionElém{; tabPositions})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
positionElém	Numérique	→	Position de l'élément dans la ou les liste(s) déployée(s)/contractée(s)
tabPositions	Tab numérique	→	Tableau de positions dans la ou les liste(s) déployée(s)/contractée(s)

Description

La commande SELECTIONNER ELEMENTS PAR POSITION sélectionne le ou les élément(s) dont vous avez passé la position dans positionElém et, facultativement, dans tabPositions, à l'intérieur de la liste dont vous avez passé le numéro de référence ou le nom d'objet dans liste.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste, vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste, la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de sa propre configuration déployée/contractée.

Note : Si vous utilisez le caractère @ dans le nom d'objet de la liste et que le formulaire contient plusieurs listes répondant à ce nom, la commande SELECTIONNER ELEMENTS PAR POSITION s'appliquera au premier objet dont le nom correspond.

La position des éléments est toujours exprimée en tenant compte de l'état déployé/contracté de la liste et de ses sous-listes. Passez des positions comprises entre 1 et la valeur retournée par Nombre elements. Si vous passez une valeur située en-dehors de cet intervalle, aucun élément n'est sélectionné.

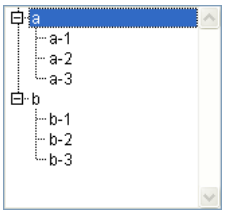
Si vous ne passez pas le paramètre tabPositions, le paramètre positionElém représente la position de l'élément à sélectionner.

Le paramètre facultatif tabPositions permet de sélectionner simultanément plusieurs éléments au sein de la liste. Vous devez passer dans tabPositions un tableau dont chaque ligne indique la position d'un élément à sélectionner.
Lorsque vous passez ce paramètre, l'élément désigné par le paramètre positionElém désigne parmi la sélection résultante le nouvel élément courant de la liste. Il peut appartenir ou non à l'ensemble d'éléments définis par le tableau. L'élément courant est notamment celui qui passe en mode édition si la commande EDITER ELEMENT est utilisée.

Note : Pour que plusieurs éléments puissent être sélectionnés simultanément dans une liste hiérarchique (manuellement ou par programmation), la propriété multiSélection doit avoir été activée pour cette liste. Cette propriété est définie via la commande CHANGER PROPRIETES LISTE.

Exemples

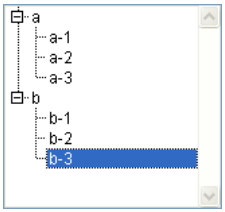
Soit une liste hiérarchique nommée hList affichée en mode Application :



(1) Après l'exécution des lignes de code suivantes :

SELECTIONNER ELEMENTS PAR POSITION(hList;**Nombre elements**(hList))

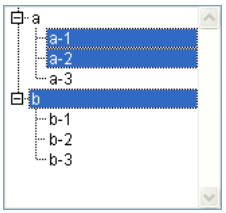
... le dernier élément visible est sélectionné :



(2) Après l'exécution des lignes de code suivantes :

```
CHANGER PROPRIETES LISTE(hList;0;0;18;0;1)
    `Il est impératif de passer 1 en dernier paramètre pour autoriser les multi-sélections
TABLEAU ENTIER LONG($tab;3)
$tab{1}:=2
$tab{2}:=3
$tab{3}:=5
SELECTIONNER ELEMENTS PAR POSITION(hList;3;$tab)
    `Le 3e élément est désigné comme élément courant
```

... les 2e, 3e et 5e éléments de la liste hiérarchique sont sélectionnés :



Référence

EDITER ELEMENT, Elements selectionnes, SELECTIONNER ELEMENTS PAR REFERENCE.

SELECTIONNER ELEMENTS PAR REFERENCE

version 2004 (Modifiée)

SELECTIONNER ELEMENTS PAR REFERENCE (liste; réfElément{; tabRéfs})

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de liste
réfElément	Entier long	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste
tabRéfs	Tab entier long	→	Tableau de numéros de référence d'éléments

Description

La commande SELECTIONNER ELEMENTS PAR REFERENCE sélectionne le ou les élément(s) dont vous avez passé le numéro de référence dans réfElément et, facultativement, dans tabRéfs, parmi la liste dont vous avez passé la référence dans liste.

Si un élément n'est pas visible (car il est par exemple inclus dans une liste contractée), SELECTIONNER ELEMENTS PAR REFERENCE déploie la ou les sous-liste(s) correspondante(s) de manière à ce qu'il devienne visible.

Si vous ne passez pas le paramètre tabRéfs, le paramètre réfElément représente la référence de l'élément à sélectionner. Si le numéro d'élément ne correspond à aucun élément de la liste, la commande ne fait rien. Vous pouvez également passer la valeur 0 dans ce paramètre afin de désigner le dernier élément ajouté à la liste.

Le paramètre facultatif tabRéfs permet de sélectionner simultanément plusieurs éléments au sein de la liste. Vous devez passer dans tabRéfs un tableau dont chaque ligne indique la référence absolue d'un élément à sélectionner.
Dans ce cas, l'élément désigné par le paramètre refElém désigne parmi la sélection résultante le nouvel élément courant de la liste. Il peut appartenir ou non à l'ensemble d'éléments définis par le tableau. L'élément courant est notamment celui qui passe en mode édition si la commande EDITER ELEMENT est utilisée.

Note : Pour que plusieurs éléments puissent être sélectionnés simultanément dans une liste hiérarchique (manuellement ou par programmation), la propriété multiSélection doit avoir été activée pour cette liste. Cette propriété est définie via la commande CHANGER PROPRIETES LISTE.

Lorsque vous travaillez avec les numéros de référence des éléments, assurez-vous d'utiliser des numéros uniques, sinon vous ne pourrez pas différencier les éléments. Pour plus d'informations sur ce point, reportez-vous à la description de la commande AJOUTER A LISTE.

Exemple

En supposant que hList est une liste dont les éléments ont des numéros de référence uniques, la méthode objet de bouton suivante sélectionne l'élément parent (s'il existe) de l'élément actuellement sélectionné :

```
$vlElémPos:=Elements selectionnes(hList)  ` Récupérer la position de l'élément sélectionné
INFORMATION ELEMENT(hList;$vlElémPos;$vlElémRef;$vsElémText)
    ` Numéro de référence de cet élément
$vlParentElémRef:=Element parent(hList;$vlElémRef)
    ` Numéro de l'élément parent (s'il existe)
Si ($vlParentElémRef>0)
        ` Sélection de l'élément parent
    SELECTIONNER ELEMENTS PAR REFERENCE(hList;Element parent(hList;$vlElémRef))
Fin de si
```

Référence

EDITER ELEMENT, Elements selectionnes, SELECTIONNER ELEMENTS PAR POSITION.

STOCKER LISTE

STOCKER LISTE (liste; nomListe)

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de liste
nomListe	Alpha	→	Nom de la liste tel qu'il doit apparaître dans l'éditeur d'énumérations en mode Développement

Description

La commande STOCKER LISTE sauvegarde la liste dont vous avez passé le numéro de référence dans liste, sous le nom que vous avez passé dans nomListe. La liste est stockée en tant qu'énumération dans l'éditeur d'énumérations du mode Développement.

Si une énumération de même nom existe déjà, son contenu est remplacé.

Référence

Charger liste.

SUPPRIMER DANS LISTE ({*; }liste; réfElément | *{; *})

Paramètre	Type		Description
*	*	→	Si spécifié, liste est un nom d'objet (chaîne) Si omis, liste est une référence de liste
liste	RéfListe | Chaîne	→	Numéro de référence de liste (si * omis) ou Nom d'objet de type liste (si * passé)
réfElément | *	Entier long | *	→	Numéro de référence d'élément ou 0 pour le dernier élément ajouté à la liste ou * pour l'élément de la liste actuellement sélectionné
*		→	Si spécifié, effacer les sous-listes de la mémoire (le cas échéant) Si omis, ne pas effacer les sous-listes

Description

La commande SUPPRIMER DANS LISTE supprime l'élément désigné par le paramètre réfElément de la liste dont le numéro de référence ou le nom d'objet est passé dans liste.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre liste est un nom d'objet (chaîne) correspondant à une représentation de liste dans le formulaire. Si vous ne passez pas ce paramètre, vous indiquez que le paramètre liste est une référence de liste hiérarchique (RéfListe). Si vous utilisez une seule représentation de liste ou travaillez avec les éléments structurels (le second * est omis), vous pouvez utiliser indifféremment l'une ou l'autre syntaxe. En revanche, si vous utilisez plusieurs représentations d'une même liste et travaillez avec l'élément courant (le second * est passé), la syntaxe basée sur le nom d'objet est requise car chaque représentation peut disposer de son propre élément courant.

Si vous passez * dans réfElément, vous supprimez l'élément actuellement sélectionné de la liste. Vous pouvez également passer 0 dans ce paramètre afin de demander la suppression du dernier élément ajouté à la liste.
Sinon, vous spécifiez le numéro de référence de l'élément à supprimer. Si le numéro ne correspond à aucun élément de la liste, la commande ne fait rien.

Si vous travaillez avec les numéros de référence des éléments, veillez à construire une liste dans laquelle les éléments ont des numéros de référence uniques, sinon vous ne pourrez les différencier. Pour plus d'informations sur ce point, reportez-vous à la description de la commande AJOUTER A LISTE.

Quel que soit l'élément que vous supprimez, vous pouvez passer un troisième paramètre optionnel, *, pour indiquer à 4D de supprimer automatiquement de la mémoire la sous-liste rattachée à l'élément, s'il en existe. Si vous ne passez pas ce paramètre, il est préférable de récupérer au préalable le numéro de référence de la sous-liste (éventuelle) rattachée à l'élément, de manière à pouvoir si besoin est supprimer cette sous-liste à l'aide de la commande SUPPRIMER LISTE.

Exemple

L'exemple suivant supprime l'élément sélectionné de la liste hList. Si une sous-liste est rattachée à l'élément, elle est supprimée (ainsi que toute sous-sous-liste) :

```
SUPPRIMER DANS LISTE(hList;*;*)
```

Référence

INFORMATION ELEMENT, SUPPRIMER LISTE.

SUPPRIMER LISTE (liste{; *})

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de liste
*		→	Si spécifié, effacer les sous-listes de la mémoire (s'il existe des sous-listes) Si omis, ne pas effacer les sous-listes

Description

La commande SUPPRIMER LISTE efface de la mémoire la liste hiérarchique dont vous avez passé le numéro de référence dans le paramètre liste.

Généralement, vous devez passer le paramètre optionnel *, afin que les sous-listes et les sous-éléments (s'il y en a) rattachés à la liste soient également effacés.

Il n'est pas nécessaire de supprimer une liste associée à un objet de formulaire via la Liste des propriétés : 4D charge et efface la liste automatiquement. Sinon, à chaque fois que vous chargez, copiez, extrayez d'un BLOB ou créez une liste par programmation, appelez la commande SUPPRIMER LISTE lorsque vous n'en avez plus besoin.

Si vous voulez supprimer une sous-liste rattachée à un élément (à tout niveau) d'une autre liste affichée dans un formulaire, procédez de la manière suivante :
1. Appelez INFORMATION ELEMENT avec l'élément parent pour obtenir le numéro de référence de la sous-liste.
2. Appelez CHANGER ELEMENT avec l'élément parent pour dissocier la sous-liste de l'élément de liste avant de l'effacer.
3. Appelez SUPPRIMER LISTE pour effacer la sous-liste dont vous avez obtenu le numéro de référence à l'aide de INFORMATION ELEMENT.

Exemples

(1) Vous disposez, dans votre application, d'une routine de "nettoyage" chargée d'effacer tous les objets et données dont vous n'avez plus besoin lorsque, par exemple, une fenêtre ou un formulaire est refermé(e). A un endroit de cette routine, vous supprimez une liste hiérarchique qui peut avoir déjà été supprimée, suivant les actions de l'utilisateur dans le formulaire. Vous utilisez la fonction Liste existante pour effacer la liste uniquement si c'est nécessaire :

```
    ` Extrait de la sous-routine de nettoyage
Si (Liste existante(hlList))
    SUPPRIMER LISTE(hlList;*)
Fin de si
```

(2) Reportez-vous à l'exemple de la fonction Charger liste.

(3) Reportez-vous à l'exemple de la fonction BLOB vers liste.

Référence

BLOB vers liste, Charger liste, Nouvelle liste.

TRIER LISTE (liste{; > ou <})

Paramètre	Type		Description
liste	RéfListe	→	Numéro de référence de liste
> ou <		→	Ordre de tri : > pour trier la liste dans l'ordre croissant ou < pour trier la liste dans l'ordre décroissant

Description

La commande TRIER LISTE effectue un tri sur la liste dont vous avez passé le numéro de référence dans le paramètre liste.

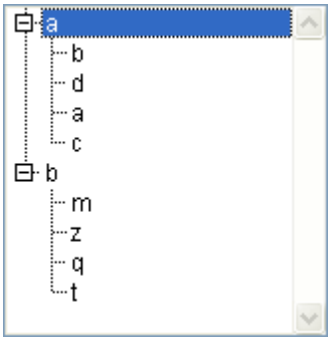
Pour effectuer un tri dans l'ordre croissant, passez > comme deuxième paramètre. Pour effectuer un tri dans l'ordre décroissant, passez < comme deuxième paramètre. Si vous omettez ce paramètre, TRIER LISTE effectue par défaut un tri croissant.

TRIER LISTE trie tous les niveaux de la liste : les éléments de la liste, puis les sous-éléments de chaque sous-liste, puis des sous-listes suivantes, etc., sont triés. C'est pourquoi généralement vous utiliserez la commande TRIER LISTE avec une liste affichée dans un formulaire. Le tri d'une sous-liste a moins d'intérêt car son ordre sera modifié dès qu'un appel à une liste se produira à un niveau supérieur.

TRIER LISTE ne modifie pas l'état courant déployé/contracté de la liste et de ses éventuelles sous-listes, ni l'élément courant. Cependant, comme l'élément courant peut être déplacé à la suite du tri, Elements selectionnes peut retourner une position différente avant et après le tri.

Exemple

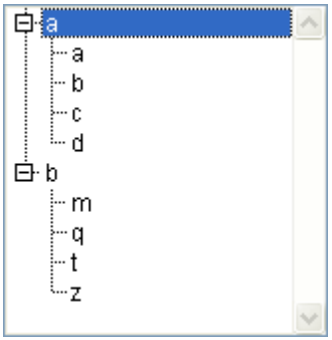
Voici la liste nommée hList, affichée ici en mode Application :



Après l'exécution du code suivant :

```
` Trier la liste et ses sous-listes dans l'ordre croissant
TRIER LISTE(hList;>)
```

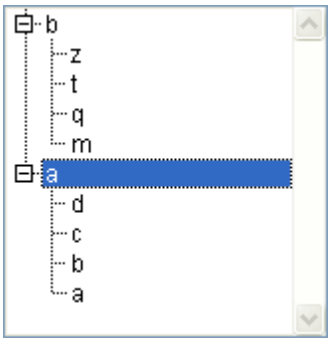
... la liste apparaît ainsi :



Après l'exécution du code suivant :

```
` Trier la liste et ses sous-listes dans l'ordre décroissant
TRIER LISTE(hList;<)
```

... la liste apparaît ainsi :



Référence

Elements selectionnes.

38

Menus

Terminologie : La documentation sur les commandes de menus emploie indifféremment **commande de menu** et **ligne de menu** lorsqu'elle évoque une ligne d'un menu.

RefMenu et numéros de menus

Le langage de 4D propose deux modes de manipulation des menus et des barres de menus : par des **références** ou des **numéros**.
• La gestion de menus par **référence** (*RefMenu*) est le nouveau mode de gestion des menus, introduit depuis la version 11 de 4D. Ce mode donne accès à des fonctions avancées telles que la création d'interfaces entièrement dynamiques (menus créés "à la volée" sans devoir exister dans l'éditeur de menus) et la gestion de sous-menus hiérarchiques multi-niveaux.
• La gestion de menus et de barres de menus par **numéro** s'appuie sur les menus créés dans l'éditeur de menus en mode Développement. Chaque barre de menus et menu se voit attribuer un numéro fixe (correspondant à sa position dans l'éditeur). Ce numéro est utilisé par les commandes du langage pour désigner la barre ou le menu. La portée des commandes du langage appliquées aux menus gérés par numéro est la barre de menus courante.
Ce fonctionnement correspond aux versions précédentes de 4D et obéit à plusieurs règles (décrites ci-dessous dans le paragraphe "Manipulation des menus par numéros"). Il peut toujours être utilisé mais ne permet pas de tirer parti des nouvelles fonctions proposées à partir de la version 11, notamment la gestion dynamique des menus et l'utilisation de sous-menus hiérarchiques : il n'est pas possible d'accéder à un sous-menu hiérarchique par un numéro.

Les deux modes de gestion des menus sont compatibles et peuvent être utilisés simultanément dans vos interfaces. La plupart des commandes du thème “Menus” acceptent indifféremment des numéros ou des références de menus.
Toutefois, la gestion par référence est conseillée car elle offre davantage de possibilités. A noter que si votre interface de menus est définie partiellement ou en totalité via l'éditeur de menus, il est parfaitement possible de l'exploiter sous forme de références à l'aide des commandes Lire reference barre menu et LIRE LIGNES MENU.

Manipulation des menus par référence

Lorsque les menus sont manipulés par l'intermédiaire de références *RefMenu*, il n’y a pas de différence de nature entre un menu et une barre de menus. Il s’agit dans les deux cas de listes de libellés. Seul leur usage diffère. Dans le cas d’une barre, chaque libellé correspond à un menu, composé de libellés. C’est également sur ce principe que repose la définition de menus hiérarchiques : chaque libellé peut à son tour être un menu, et ainsi de suite.

Lorsqu'un menu est géré par référence, toute modification effectuée sur ce menu durant la session est immédiatement répercutée à l'ensemble des instances de ce menu et ce, dans tous les process de la base.

RefMenu
A l'image des listes hiérarchiques, tous les menus disposent d'une référence unique, grâce à laquelle il pourront être identifiés durant toute la session. Cette référence, nommée par convention RefMenu, est un alphanumérique de 16 caractères. Toutes les commandes du thème "Menus" acceptent soit cette référence, soit un numéro de menu pour désigner un menu ou une barre.
Les références des menus peuvent être obtenues par les commandes Creer menu, Lire reference barre menu ou LIRE LIGNES MENU.

Manipulation des menus par numéro

Barres de menus
Les barres de menus peuvent être définies dans l'éditeur de menus en mode Développement. En mode de gestion par numéro, chaque barre de menus est identifiée par un numéro et par un nom. La première barre de menus (automatiquement créée par 4D) porte le numéro 1 et est nommée par défaut "Barre n°1". Vous pouvez la renommer dans l'éditeur de menu. Le nom d'une barre de menus peut contenir jusqu'à 31 caractères et doit être unique.
La barre n°1 est aussi la barre de menus utilisée par défaut. Si vous souhaitez ouvrir une application avec une barre de menus autre que la barre n°1, vous devez appeler la commande FIXER BARRE MENUS dans la Méthode base Sur ouverture. Il n'est pas possible de modifier par programmation le contenu même d'une barre de menus, en revanche les menus qui la composent peuvent être modifiés. La portée des commandes du langage appliquées aux menus statiques est la barre de menus courante. A chaque appel de la commande FIXER BARRE MENUS (sans le paramètre *), tous les menus et les commandes de menus retrouvent leur état originel tel que défini dans l'éditeur de menus.

Chaque barre de menus comporte par défaut trois menus — **Fichier**, **Edition** et **Mode**.
• Le menu **Fichier** ne contient qu'une commande de menu : **Quitter**. L'action standard Quitter lui est associée. Cette action affiche une boîte de dialogue de confirmation "Etes-vous certain ?" puis quitte l'application 4D en cas de validation. Dans le cas contraire, l'opération est annulée.

Note : Sous Mac OS, la commande de menu créé associée à l'action Quitter est automatiquement placée dans le menu de l'application, lorsque la base est exécutée sur ce système.

Vous pouvez renommer le menu Fichier, lui ajouter des commandes de menu ou le garder tel quel. Il est recommandé de toujours garder la commande de menu Quitter comme dernière commande du menu Fichier.

• Le menu **Edition** contient les commandes de menu d'édition standard. A chaque commande de ce menu est associée une action standard (Annuler, Couper, Copier, etc.). Vous pouvez ajouter des commandes à ce menu ou utiliser vos propres méthodes de gestion des actions d'édition.
• Le menu **Mode** contient par défaut la commande Retour au mode Développement. Cette commande permet de retourner au mode Développement (lorsqu'il est disponible) à partir du mode Application.

Note : 4D gère automatiquement les menus système **Aide** et **application** (Mac OS). Ces menus ne peuvent pas être modifiés, hormis pour la commande **A propos de 4D...,** qui peut être gérée à l'aide de la commande APPELER SUR A PROPOS.

Important : Les barres de menus sont "interprocess". Toute modification effectuée sur une barre en mode Développement sera répercutée dans tous les process où la barre est utilisée.

Numéros des menus et des commandes de menus
Comme les barres de menus, les menus sont numérotés. Le menu Fichier est généralement le menu 1. Les autres menus sont numérotés séquentiellement de gauche à droite (2, 3, 4, etc.). Le menu **Application** (Mac OS) est exclu de cette numérotation. Sur toutes les plates-formes, le menu **Aide** est également exclu. Il est à noter que la commande Nombre de menus ne tient pas compte de ces menus. Si, par exemple, votre barre de menus est constituée des menus Fichier, Edition, Clients, Factures et Aide, Nombre de menus retournera 4 (en ignorant les menus système maintenus par 4D).
La numérotation des menus est importante lorsque vous travaillez, par exemple, avec la fonction Menu choisi.

Lorsqu'un menu est associé à un formulaire, le principe de numérotation est différent. Le premier menu ajouté commence avec le numéro 2049. Pour référencer un menu associé à un formulaire, ajoutez 2048 au numéro initial du menu.

Les commandes de chacun des menus sont numérotées séquentiellement de haut en bas, y compris les séparateurs. La commande supérieure a le numéro 1.

Barres de menus associées
Vous pouvez associer une barre de menus à un formulaire dans les Propriétés du formulaire (page Général). Ce type de barre est appelé "barre de menus de formulaire" dans cette section.
Les menus d'une barre de menus de formulaire sont ajoutés à la barre de menus courante lorsque le formulaire est affiché comme formulaire de sortie dans le mode Application.
Les barres de menus de formulaires sont référencées par un numéro et un nom. Si le numéro ou le nom de la barre de menus affichée avec le formulaire courant est le même que celui de la barre de menus associée au formulaire, cette dernière ne s'affiche pas.
Par défaut, lorsqu'un formulaire est affiché avec une barre de menus personnalisée, les commandes de la barre de menus courante sont inactivées, c'est-à-dire que leur sélection est sans effet. Vous pouvez modifier ce fonctionnement en cochant l'option **Barre de menus active** dans les Propriétés du formulaire : dans ce cas, les commandes de la barre de menus courante restent utilisables.

Dans tous les cas, la sélection d'une commande de menu provoque l'envoi d'un événement Sur menu sélectionné à la méthode formulaire ; vous pouvez alors utiliser la commande Menu choisi pour tester le menu sélectionné.

Menus rattachés
Les menus peuvent être rattachés à des barres de menus. Si un menu rattaché est modifié à l'aide d'une de ces commandes, chacune des instances de ce menu reflètera ces modifications. Pour plus d'informations sur ce point, reportez-vous au manuel *Mode Développement* de 4D.

Actions standard et méthodes associées aux commandes de menus

Chaque commande de menu peut être associée à une méthode projet ou une action standard. Si vous n'affectez pas de méthode ni d'action standard à une commande de menu, la sélection de cette commande de menu provoque la sortie du mode Application et le retour en mode Développement. Si seul le mode Application est disponible ou si l'utilisateur ne dispose pas des privilèges d'accès pour le mode Développement, cela provoquera la fermeture de l'application.
Les actions standard permettent d'effectuer diverses opérations courantes liées aux fonctions système (copier, quitter, etc.) ou de base de données 4D (ajouter enregistrement, tout sélectionner, etc.).

Vous pouvez associer à la fois une action standard et une méthode projet à une commande de menu. Dans ce cas, l'action standard n'est jamais exécutée ; toutefois, 4D utilise cette action pour activer/inactiver la commande de menu en fonction du contexte et lui associer éventuellement un comportement spécifique en fonction de la plate-forme (par exemple, l'action Préférences est passée dans le menu application sous Mac OS). Lorsqu'une commande de menu est inactivée, la méthode projet associée ne peut être exécutée.

ligneMenu=-1

Afin de faciliter la manipulation des lignes de menus, 4D propose un raccourci permettant de désigner la dernière ligne ajoutée au menu : il suffit pour cela de passer -1 dans le paramètre ligneMenu.
Ce principe est utilisable dans toutes les commandes du thème "Menus" manipulant des lignes de menus.

ACTIVER LIGNE MENU

version 11 (Modifiée)

ACTIVER LIGNE MENU (menu; ligneMenu{; process})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence du process

Description

ACTIVER LIGNE MENU active la commande de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si le paramètre ligneMenu désigne un sous-menu hiérarchique, toutes les lignes de ce menu et de ses éventuels sous-menus sont activées. Cette commande fonctionne également avec une barre de menus créée avec la commande Creer menu et installée avec la commande FIXER BARRE MENUS.

Si vous omettez le paramètre process, ACTIVER LIGNE MENU s'applique à la barre de menus du process courant. Sinon, ACTIVER LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Astuce : Pour activer/inactiver toutes les lignes de menus en une fois, passez 0 (zéro) dans ligneMenu.

Référence

INACTIVER LIGNE MENU.

AJOUTER LIGNE MENU

version 11.4 (Modifiée)

AJOUTER LIGNE MENU (menu; libelléLigne{; sousMenu{; process{; *}}})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
libelléLigne	Texte	→	Libellé du ou des nouvelle(s) ligne(s) de menu
sousMenu	RefMenu	→	Référence du sous-menu associé à la ligne
process	Numérique	→	Numéro de référence du process
*	*	→	Si passé : considérer les métacaractères comme des caractères standard

Description

La commande AJOUTER LIGNE MENU ajoute une ou plusieurs ligne(s) au menu dont vous avez passé le numéro ou la référence dans menu.

Si vous omettez le paramètre process, AJOUTER LIGNE MENU s'applique à la barre de menus du process courant. Sinon, AJOUTER LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Si vous ne passez pas le paramètre *, AJOUTER LIGNE MENU vous permet d'ajouter une ou plusieurs lignes de menu en un seul appel. Vous définissez les lignes à ajouter à l'aide du paramètre libelléLigne, de la manière suivante :

- Chaque ligne est séparée des autres par un point-virgule ";", "ligne1;ligne2;ligne3".
- Pour inactiver une ligne, placez une parenthèse ouvrante "(" dans son libellé.
- Pour définir une ligne de séparation, passez la valeur "(-" en tant que libellé.
- Pour définir le style de caractères d'une ligne, placez dans son libellé le symbole inférieur à "<" suivi d'une lettre. Voici les différents codes :

<B	Gras
<I	Italique
<U	Souligné

- Pour associer une coche à une ligne, insérez dans son libellé un point d'exclamation "!" suivi du caractère que vous voulez utiliser comme coche. Sous Mac OS, le caractère est affiché ; sous Windows, une coche standard est affichée (quel que soit le caractère passé).

• Pour associer une icône à une ligne, insérez dans son libellé un accent circonflexe "^" suivi d'un caractère dont le code moins 48 plus 256 (ou plus 208) représente un numéro de ressource d'icône Mac OS.
• Pour ajouter un raccourci clavier à une ligne, insérez dans son libellé une barre oblique "/" suivie du caractère de raccourci.

Note : Utilisez les menus avec un nombre "raisonnable" de lignes. Si, par exemple, vous voulez afficher plus de 50 lignes, envisagez plutôt d'employer une zone de défilement dans un formulaire.

Si vous passez le paramètre *, les caractères "spéciaux" inclus dans les libellés des lignes (; (!...) seront considérés comme des caractères standard et non comme des métacaractères. Ce principe vous permet de créer des lignes avec un libellé tel que "Copier (spécial)..." ou "Chercher/Remplacer...". A noter que lorsque le paramètre * est passé, vous ne pouvez pas créer plusieurs lignes en un seul appel, le caractère ";" étant considéré comme un caractère standard.

Note : Les commandes LIRE LIGNES MENU et Lire texte ligne menu retourneront ou non les métacaractères d'un libellé en fonction de son mode de création : s'il a été créé avec l'option *, les métacaractères seront retournés en tant que caractères standard.

Le paramètre facultatif sousMenu vous permet de désigner un menu comme ligne ajoutée et donc de définir un sous-menu hiérarchique. Vous devez passer dans ce paramètre une référence de menu (chaîne de type *RefMenu*) désignant un menu créé par exemple à l'aide de la commande Creer menu. Si la commande ajoute plusieurs lignes de menus, le sous-menu est associé à la première ligne.

Important : Les nouvelles lignes n'ont pas de méthodes ou d'actions associées. Vous devez leur associer une action ou une méthode via les commandes FIXER PROPRIETE LIGNE MENU ou FIXER METHODE LIGNE MENU ou encore les gérer à partir d'une méthode formulaire qui utilise la fonction Menu choisi.

Exemple

L'exemple suivant ajoute les noms des polices de caractères disponibles dans un menu **Polices** qui, dans cet exemple, est le sixième menu de la barre de menus courante :

```
    ` Dans la méthode base Sur ouverture
    ` La liste des polices est chargée et les libellés construits
LISTE DES POLICES(<>asPolicesDispo)
<>atPoliceCmdMenus:=""
Boucle ($vlPolice;1;Taille tableau(<>asPolicesDispo))
    <>atPoliceCmdMenus:=<>atPoliceCmdMenus+";"+<>asPolicesDispo{$vlPolice}
Fin de boucle
```

Ensuite, dans toute méthode formulaire ou projet, vous pouvez écrire :

```
AJOUTER LIGNE MENU(6;<>atPoliceCmdMenus)
```

Référence

FIXER PARAMETRE LIGNE MENU, INSERER LIGNE MENU, SUPPRIMER LIGNE MENU.

Creer menu {(menu)} → RefMenu

Paramètre	Type		Description
menu	RefMenu | Num | Alpha	→	Référence de menu ou Numéro ou Nom de barre de menus
Résultat	RefMenu	←	Référence du menu

Description

La commande Creer menu permet de créer un nouveau menu en mémoire. Ce menu n'existera qu'en mémoire et ne sera pas ajouté dans l'éditeur de menus en mode Développement. Toute modification effectuée sur ce menu durant la session sera immédiatement répercutée à l'ensemble des instances de ce menu et ce, dans tous les process de la base.

La commande retourne un identifiant unique de type RefMenu pour le nouveau menu.

• Si vous ne passez pas le paramètre facultatif menu, le menu sera créé vide. Vous devrez le construire et le gérer à l'aide des commandes AJOUTER LIGNE MENU, FIXER TEXTE LIGNE MENU, etc.
• Si vous passez le paramètre menu, le menu créé sera une copie exacte du menu source désigné par ce paramètre. Toutes les propriétés du menu source, y compris les éventuels sous-menus associés, seront appliquées au nouveau menu. A noter qu'une nouvelle référence RefMenu est créée pour le menu source et pour chaque sous-menu associé existant.

Vous pouvez passer dans menu soit une référence de menu valide, soit un numéro ou un nom de barre de menus défini en mode Développement. Dans ce dernier cas, le nouveau menu sera constitué des menus et sous-menus de la barre d'origine.

Un menu créé par cette commande peut être utilisé en tant que barre de menus à l'aide de la commande FIXER BARRE MENUS.

Exemple

Reportez-vous à l'exemple de la commande FIXER BARRE MENUS.

Référence

EFFACER MENU, FIXER BARRE MENUS, Pop up menu dynamique.

EFFACER MENU (menu)

Paramètre	Type		Description
menu	RefMenu	→	Référence de menu

Description

La commande EFFACER MENU efface de la mémoire le menu dont vous avez passé l'identifiant dans menu. Ce menu doit avoir été créé par la commande Creer menu.

La commande efface toutes les instances du menu dans toutes les barres de menus et tous les process. Si le menu appartient à une barre de menus en cours d'utilisation, il continuera à fonctionner mais ne pourra plus être modifié. Il ne sera réellement effacé de la mémoire que lorsque la dernière barre de menus dans laquelle il figure ne sera plus utilisée.

Cette commande peut être appliquée aux menus utilisés comme barres de menus. Les sous-menus éventuellement utilisés par menu ne sont pas effacés. Si vous souhaitez effacer un menu et ses sous-menus, vous devez effacer chaque sous-menu individuellement.

Référence

Creer menu.

FIXER BARRE MENUS

version 11 (Modifiée)

FIXER BARRE MENUS (barre{; process}{; *})

Paramètre	Type		Description
barre	Num | Alpha | RefMenu	→	Numéro ou nom de la barre de menus ou Référence de menu
process	Numérique	→	Numéro de référence du process
*		→	Conserver l'état de la barre de menus

Description

La commande FIXER BARRE MENUS remplace la barre de menus courante par la barre de menus barre, pour le process en cours uniquement. Vous pouvez passer dans le paramètre barre soit le numéro soit le nom de la nouvelle barre. Vous pouvez également passer une référence unique de menu (type RefMenu, chaîne de 16 caractères). Lorsque vous travaillez avec des références, les menus peuvent être utilisés comme barres de menus et inversement (cf. section Gestion des menus).

Note : Le nom d'une barre de menus peut contenir jusqu'à 31 caractères et doit être unique.

Si vous passez le paramètre optionnel process, c'est la barre de menus du process spécifié qui sera remplacée par la barre.

Note : Si vous passez un paramètre RefMenu dans barre, le paramètre process est inutile et sera ignoré.

Le paramètre optionnel * vous permet de conserver l'état de la barre de menus. Si ce paramètre est omis, FIXER BARRE MENUS réinitialise la barre de menus lors de l'exécution de la commande. Imaginez, par exemple, que l'instruction FIXER BARRE MENUS(1) soit exécutée. Ensuite, plusieurs commandes de menu sont désactivées à l'aide de la commande INACTIVER LIGNE MENU.
Si FIXER BARRE MENUS(1) est exécutée une seconde fois, soit à partir du même process, soit à partir d'un autre process, toutes les commandes de menu retournent à leur état d'activation initial.
Si FIXER BARRE MENUS(1;*) est exécutée, la barre de menus conservera son état précédent, les commandes de menu qui étaient inactivées le resteront.

Note : Si vous passez un paramètre RefMenu dans barre, le paramètre * est inutile et sera ignoré.

Lorsqu'un utilisateur arrive en mode Application, la première barre de menus s'affiche (Barre n° 1). Vous pouvez changer cette barre de menus par défaut en spécifiant la barre que vous voulez dans la Méthode base Sur ouverture, ou dans la méthode de démarrage associée à un utilisateur.

Exemples

(1) L'exemple suivant remplace la barre de menus courante par la barre de menus n° 3 et initialise l'état des commandes des menus :

```
FIXER BARRE MENUS (3)
```

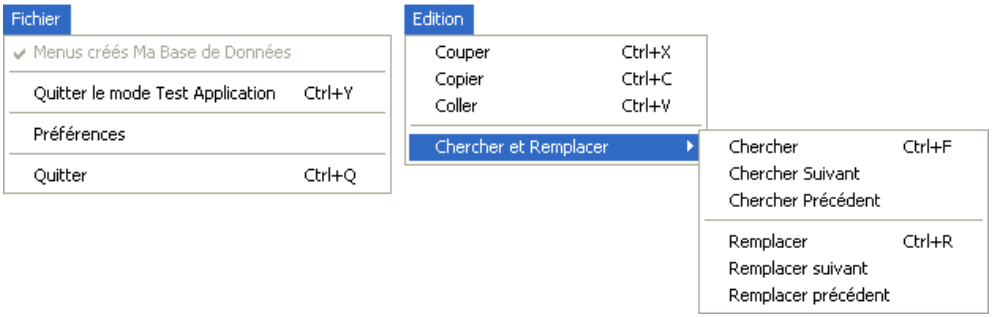
(2) L'exemple suivant remplace la barre de menus courante par la barre de menus nommée “BarreForm1” et conserve l'état des commandes des menus : celles qui étaient précédemment inactivées apparaîtront inactivées :

```
FIXER BARRE MENUS ("BarreForm1";*)
```

(3) L'exemple suivant remplace la barre de menus courante par la barre de menus n° 3 pendant que des enregistrements sont en cours de modification. Une fois les enregistrements modifiés, la barre de menus n° 2 est réaffichée. L'état des commandes de ce menu est conservé :

```
FIXER BARRE MENUS(3)  ` Définir la barre de menus n° 3 pour le formulaire suivant
TOUT SELECTIONNER([Clients])
MODIFIER SELECTION([Clients])  ` Afficher la sélection
FIXER BARRE MENUS(2;*)  ` Après modification, retour à la barre de menus n° 2
```

(4) Dans cet exemple complet, nous allons créer par programmation une barre comportant les menus Fichier et Edition suivants :



```
	`Méthode de création menu Fichier
C_ALPHA(16;FileMenu)  ` FileMenu contiendra la référence du menu Fichier
FileMenu:=Creer menu
INSERER LIGNE MENU(FileMenu;-1;Lire chaine dans liste(131;29)+
                                                    " Ma Base de Données(")
FIXER MARQUE LIGNE MENU(FileMenu;1;Caractere(18))
INSERER LIGNE MENU(FileMenu;-1;"(-")
INSERER LIGNE MENU(FileMenu;-1;"Quitter le mode Test Application/Y")
FIXER PROPRIETE LIGNE MENU(FileMenu;3;Action standard associée;
                                                    Retour au mode Développement)
INSERER LIGNE MENU(FileMenu;-1;"(-")
INSERER LIGNE MENU(FileMenu;-1;Lire chaine dans liste(131;26))
	`Preferences
FIXER PROPRIETE LIGNE MENU(FileMenu;6;Action standard associée; Action préférences)
INSERER LIGNE MENU(FileMenu;-1;"(-")
INSERER LIGNE MENU(FileMenu;-1;Lire chaine dans liste(131;30))
	`Quitter
FIXER PROPRIETE LIGNE MENU(FileMenu;7;Action standard associée; Action quitter)
FIXER RACCOURCI LIGNE MENU(FileMenu;7;Code de caractere("Q"))

	`Méthode de création menu Chercher et Remplacer
	`FindAndReplaceMenu contiendra la référence du menu Chercher remplacer
C_ALPHA(16;FindAndReplaceMenu)
FindAndReplaceMenu:=Creer menu
AJOUTER LIGNE MENU(FindAndReplaceMenu; "Chercher;Chercher Suivant;
            Chercher Précédent;(-;Remplacer;Remplacer suivant;Remplacer précédent")
FIXER RACCOURCI LIGNE MENU(FindAndReplaceMenu;1;Code de caractere("F"))
FIXER RACCOURCI LIGNE MENU(FindAndReplaceMenu;5;Code de caractere("R"))
FIXER METHODE LIGNE MENU(FindAndReplaceMenu;1; "MaMethodechercher")

	`Méthode de création menu Edition
C_ALPHA(16; EditMenu)  `EditMenu contiendra la référence du menu Edition
EditMenu:=Creer menu
AJOUTER LIGNE MENU(EditMenu;"Couper;Copier;Coller")
FIXER RACCOURCI LIGNE MENU(EditMenu;1;Code de caractere("X"))
FIXER PROPRIETE LIGNE MENU(EditMenu;1;Action standard associée; Action couper )
FIXER RACCOURCI LIGNE MENU(EditMenu;2;Code de caractere("C"))
FIXER PROPRIETE LIGNE MENU(EditMenu;2;Action standard associée;Action copier )
FIXER RACCOURCI LIGNE MENU(EditMenu;3;Code de caractere("V"))
FIXER PROPRIETE LIGNE MENU(EditMenu;3;Action standard associée;Action coller )
INSERER LIGNE MENU(EditMenu;-1;"(-")
```

```
            ` ligne qui aura le sous menu
INSERER LIGNE MENU(EditMenu;-1;"Chercher et Remplacer" ; FindAndReplaceMenu)

main_Bar:=Creer menu ` Cree la barre constituée des autres menus
INSERER LIGNE MENU(main_Bar;-1;Lire chaine dans liste(79;1);FileMenu)
AJOUTER LIGNE MENU(main_Bar; "Edition";EditMenu)

FIXER BARRE MENUS(main_Bar)
```

Référence

Gestion des menus.

FIXER ICONE LIGNE MENU (menu; ligneMenu; refIcône{; process})

Paramètre	Type		Description
menu	RefMenu | Num	→	Référence de menu ou Numéro de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
refIcône	Texte | Entier long	→	Nom ou numéro de l'image à associer à la ligne
process	Numérique	→	Numéro de process

Description

La commande FIXER ICONE LIGNE MENU permet de modifier l'icône associée à la ligne de menu désignée par les paramètres menu et ligneMenu. L'icône associée à une ligne de menu est ajoutée dans la barre d'outils de l'application. L'image est affichée dans un cadre de 20 x 20 pixels.

Vous pouvez passer dans menu un identifiant unique de menu(RefMenu) ou un numéro de menu. Si vous passez un identifiant unique, la commande s'appliquera à toutes les instances du menu dans tous les process. Dans ce cas, le paramètre process est ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Passez dans le paramètre refIcône l'image devant être utilisée comme icône. Vous pouvez utiliser une image de la bibliothèque ou une référence d'image.
• Image de la bibliothèque : vous pouvez passer soit le nom soit le numéro de l' image. Il est généralement préférable d'utiliser le numéro plutôt que le nom, car les numéros d'images sont des identifiants uniques, ce qui n'est pas le cas des noms.
• Référence d'image : l'image doit se trouver dans le dossier **Resources** de la base et vous devez utiliser une syntaxe du type "file:*{cheminaccès}*nomFichier" dans refIcône. Pour plus d'informations sur le dossier **Resources**, reportez-vous à la section Ressources.

Exemple

Utilisation d'une image se trouvant dans le dossier Resources de la base :

```
FIXER ICONE LIGNE MENU($RefMenu;2;"File:French.lproj/spot.png")
```

Référence

LIRE ICONE LIGNE MENU.

FIXER MARQUE LIGNE MENU

version 11 (Modifiée)

FIXER MARQUE LIGNE MENU (menu; ligneMenu; marque{; process})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
marque	Alpha	→	Nouvelle marque de ligne de menu
process	Numérique	→	Numéro de référence du process

Description

La commande FIXER MARQUE LIGNE MENU remplace la marque (ou "coche") de la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu par le premier caractère de la chaîne que vous avez passée dans marque (sous Mac OS) ou par la coche standard (sous Windows). Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous omettez le paramètre process, FIXER MARQUE LIGNE MENU s'applique à la barre de menus du process courant. Sinon, FIXER MARQUE LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Si vous passez une chaîne vide dans marque, vous supprimez toute marque de la ligne de menu.
Sinon :
• Sous Mac OS, le premier caractère de la chaîne devient la marque de la ligne de menu (généralement, le Caractere (18), qui est la coche standard de Mac OS, est utilisé).
• Sous Windows, la marque standard de Windows est associée au menu.

Exemple

Reportez-vous à l'exemple de la commande Lire marque ligne menu.

Référence

Lire marque ligne menu.

FIXER METHODE LIGNE MENU

FIXER METHODE LIGNE MENU (menu; ligneMenu; nomMéthode{; process})

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ou Numéro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
nomMéthode	Chaîne	→	Nom de la méthode
process	Entier long	→	Numéro de process

Description

La commande FIXER METHODE LIGNE MENU permet de modifier la méthode projet 4D associée à la ligne de menu désignée par les paramètres menu et ligneMenu.
Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu) ou un numéro de menu. Si vous passez un identifiant unique, la commande s'appliquera à toutes les instances du menu dans tous les process. Dans ce cas, le paramètre process est ignoré s'il est passé. Si vous passez un numéro de menu, la commande s'appliquera au menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

Passez dans méthode le nom de la méthode 4D sous la forme d'une chaîne de caractères (expression).

Note : Si la ligne de menu correspond au libellé d'un sous-menu hiérarchique, la méthode ne sera pas appelée lorsque la ligne de menu sera sélectionnée.

Exemple

Reportez-vous aux exemple de la commande FIXER BARRE MENUS.

Référence

Lire methode ligne menu.

FIXER PROPRIETE LIGNE MENU (menu; ligneMenu; propriété; valeur{; process})

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ou Numéro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
propriété	Chaîne	→	Type de propriété
valeur	Expression	→	Valeur de la propriété
process	Entier long	→	Numéro de process

Description

La commande FIXER PROPRIETE LIGNE MENU permet de fixer la valeur de la propriété pour la ligne de menu désignée par les paramètres menu et ligneMenu.
Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu) ou un numéro de menu. Si vous passez un identifiant unique, la commande s'appliquera à toutes les instances du menu dans tous les process. Dans ce cas, le paramètre process est ignoré s'il est passé. Si vous passez un numéro de menu, la commande s'appliquera au menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

Passez dans le paramètre propriété la propriété dont vous souhaitez modifier la valeur et dans valeur, la nouvelle valeur. Pour le paramètre propriété, vous pouvez utiliser l'une des constantes du thème "Propriétés des lignes de menu" ou toute valeur personnalisée :

• Propriété standard : les constantes du thème "Propriétés des lignes de menu" ainsi que leurs valeurs possibles sont décrites ci-dessous. A noter que dans le cas de la propriété Action standard associée, vous pouvez passer une des constantes du thème "Valeurs pour actions standard associée" dans le paramètre valeur.

propriété (Constante)	valeur (Valeurs possibles)
Action standard associée Permet d'associer une action standard à la ligne de menu.	0 = Pas d'action 1 = Action ne pas valider 2 = Action valider 3 = Action enregistrement suivant 4 = Action enregistrement précédent 5 = Action premier enregistrement 6 = Action dernier enregistrement 7 = Action supprimer enregistrement 8 = Action page suivante

	9 = Action page précédente 10 = Action première page 11 = Action dernière page 12 = Action modifier sous enreg 13 = Action supprimer sous enreg 14 = Action ajouter sous enreg 17 = Action annuler 18 = Action couper 19 = Action copier 20 = Action coller 21 = Action effacer 22 = Action tout sélectionner 23 = Action afficher Presse papiers 26 = Action test application 27 = Action quitter 31 = Action répéter 32 = Action préférences 35 = Retour au mode Développement 36 = Action CSM
Démarrer un process Permet d'activer l'option "Démarrer un nouveau process".	0 = Non 1 = Oui
Autorisations d'accès Permet d'affecter un groupe d'accès à la commande.	0 = Sans restriction >0 = Numéro de groupe

Pour plus d'informations sur les propriétés standard des lignes de menus, reportez-vous au chapitre "Créer des menus personnalisés" dans le manuel *Mode Développement*.

• **Propriété personnalisée** : vous pouvez passer dans propriété tout texte personnalisé et lui associer une valeur de type texte, numérique ou booléen. Cette valeur sera stockée avec l'élément et pourra être récupérée via la commande LIRE PROPRIETE LIGNE MENU. Vous pouvez utiliser toute chaîne personnalisée dans le paramètre propriété, veillez simplement à ne pas utiliser de libellé utilisé par 4D (par convention, les propriétés définies par 4D débutent par les caractères "4D_").

Note : Si la ligne de menu correspond au libellé d'un sous-menu hiérarchique, l'action standard ne sera pas appelée lorsque la ligne de menu sera sélectionnée.

Référence

LIRE PROPRIETE LIGNE MENU.

FIXER RACCOURCI LIGNE MENU (menu; ligneMenu; touche{; modificateurs}{; process})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro du menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
touche	Num | Texte	→	Code de caractère du raccourci clavier ou Lettre du raccourci clavier
modificateurs	Entier long	→	Modificateur(s) à associer au raccourci (ignoré si un code de touche est passé)
process	Numérique	→	Numéro de référence du process

Description

La commande FIXER RACCOURCI LIGNE MENU remplace la touche du raccourci clavier associé à la ligne de menu désignée par menu et cmdeMenu, par le caractère dont vous avez passé le code de caractère ou le texte dans touche. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.
La touche définie sera automatiquement combinée à la touche **Ctrl** (Windows) ou **Commande** (Macintosh) pour définir le nouveau raccourci clavier.

Vous pouvez passer directement un nom de touche sous forme de texte (une lettre) dans le paramètre touche, par exemple “U” pour définir le raccourci **Ctrl+U** (Windows) ou **Commande+U** (Mac OS). Lorsque vous utilisez cette syntaxe, vous pouvez également passer le paramètre facultatif modificateurs afin d’associer des modificateurs additionnels au raccourci standard. Vous pouvez ainsi définir des raccourcis du type **Ctrl+Alt+Maj+Z** (Windows) ou **Cmd+Option+Maj+Z** (Mac OS).
Pour cela, passez dans modificateurs les valeurs suivantes :
• 512 pour la touche **Majuscule**
• 2048 pour la touche **Option** (Mac OS) ou **Alt** (Windows)
• Pour associer les deux touches, cumulez leurs valeurs.
A noter que les touches **Ctrl** (Windows) et **Commande** (Mac OS) sont automatiquement ajoutées par 4D au raccourci clavier.

Note : Vous pouvez définir la valeur à passer à l’aide des constantes Masque touche majuscule et Masque touche option du thème “Evénements (Modifiers)”.

Le paramètre modificateurs n'est pas pris en compte lorsque la touche de modification est définie via son code de caractère (ancienne syntaxe).

Si vous ne passez pas le paramètre process, FIXER RACCOURCI LIGNE MENU est appliquée à la barre de menus du process courant. Sinon, FIXER RACCOURCI LIGNE MENU est appliquée à la barre de menus du process dont la référence est passée dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Si vous passez 0 (zéro) dans touche, l'équivalent clavier de la commande de menu est supprimé.

Exemple

Définition du raccourci Ctrl+Maj+U (Windows) et Cmd+Maj+U (Mac OS) pour la ligne "Souligné" :

```
FIXER TEXTE LIGNE MENU(menuRef;1;"Souligné")
FIXER RACCOURCI LIGNE MENU(menuRef;1;"U";Masque touche majuscule)
```

Référence

Lire modificateurs ligne menu, Lire touche ligne menu.

FIXER PARAMETRE LIGNE MENU

version 11.4 (Modifiée)

FIXER PARAMETRE LIGNE MENU (menu; ligneMenu; param)

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ou Numéro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
param	Chaîne	→	Chaîne à associer en tant que paramètre

Description

La commande FIXER PARAMETRE LIGNE MENU vous permet d'associer une chaîne de caractères personnalisée à la ligne de menu désignée par les paramètres menu et ligneMenu.

Ce paramètre sera principalement utilisé par la commande Pop up menu dynamique.

Référence

Lire parametre ligne menu, Lire parametre ligne menu selectionnee, Pop up menu dynamique.

FIXER STYLE LIGNE MENU

version 11 (Modifiée)

FIXER STYLE LIGNE MENU (menu; ligneMenu; styleLigne{; process})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
styleLigne	Numérique	→	Nouveau style de la ligne de menu
process	Numérique	→	Numéro de référence du process

Description

La commande FIXER STYLE LIGNE MENU remplace le style de police de la ligne de menu, dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu, par le style de police que vous avez passé dans styleLigne. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous omettez le paramètre process, FIXER STYLE LIGNE MENU s'applique à la barre de menus du process courant. Sinon, FIXER STYLE LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Vous pouvez définir le style de l'élément dans le paramètre styleLigne. Vous passez une ou une combinaison des constantes prédéfinies suivantes, placées dans le thème Styles de caractères :

Constante	Type	Valeur
Standard	Entier long	0
Gras	Entier long	1
Italique	Entier long	2
Souligné	Entier long	4

Référence

Lire style ligne menu.

FIXER TEXTE LIGNE MENU (menu; ligneMenu; texteLigne{; process}{; *})

Paramètre	Type		Description
menu	Num | RefMEnu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
texteLigne	Alpha	→	Nouveau libellé de la ligne de menu
process	Numérique	→	Numéro de référence de process
*	*	→	Si passé : considérer les métacaractères comme des caractères standard

Description

La commande FIXER TEXTE LIGNE MENU remplace le libellé de la ligne de menu, dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu, par le libellé que vous avez passé dans texteLigne. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous ne passez pas le paramètre *, les caractères "spéciaux" inclus dans texteLigne (tels que (; ou !) seront considérés comme des caractères d'instruction (métacaractères). Par exemple, pour définir une ligne de menu comme ligne de séparation, insérez le caractère “-” dans texteLigne. Si vous passez le paramètre *, les caractères "spéciaux" seront considérés comme des caractères standard. Reportez-vous à la description de la commande AJOUTER LIGNE MENU pour plus de détails sur ces caractères.

Si vous omettez le paramètre process, FIXER TEXTE LIGNE MENU s'applique à la barre de menus du process courant. Sinon, FIXER TEXTE LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Référence

AJOUTER LIGNE MENU, FIXER RACCOURCI LIGNE MENU, Lire texte ligne menu.

INACTIVER LIGNE MENU

version 11 (Modifiée)

INACTIVER LIGNE MENU (menu; ligneMenu{; process})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence du process

Description

INACTIVER LIGNE MENU désactive la commande de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si le paramètre ligneMenu désigne un sous-menu hiérarchique, toutes les lignes de ce menu et de ses éventuels sous-menus sont inactivées. Cette commande fonctionne également avec une barre de menus créée avec la commande Creer menu et installée avec la commande FIXER BARRE MENUS.

Si vous omettez le paramètre process, INACTIVER LIGNE MENU s'applique à la barre de menus du process courant. Sinon, INACTIVER LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Astuce : Pour activer/inactiver toutes les lignes de menus en une fois, passez 0 (zéro) dans ligneMenu.

Référence

ACTIVER LIGNE MENU.

INSERER LIGNE MENU (menu; aprèsLigne; libelléLigne{; sousMenu{; process{; *}}})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
aprèsLigne	Numérique	→	Numéro de commande de menu
libelléLigne	Alpha	→	Libellé de la ligne de menu à insérer
sousMenu	RefMenu	→	Référence du sous-menu associé à la ligne
process	Numérique	→	Numéro de référence de process
*	*	→	Si passé : considérer les métacaractères comme des caractères standard

Description

La commande INSERER LIGNE MENU insère de nouvelles lignes dans le menu dont vous avez passé le numéro ou la référence dans menu et les place après la ligne de menu dont le numéro est passé dans aprèsLigne.

Si vous ne passez pas le paramètre process, INSERER LIGNE MENU est appliquée à la barre de menus du process courant. Sinon, INSERER LIGNE MENU est appliquée à la barre de menus du process dont la référence est passée dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Si vous ne passez pas le paramètre *, INSERER LIGNE MENU vous permet d'insérer une ou plusieurs lignes de menus en une seule fois.

INSERER LIGNE MENU fonctionne comme AJOUTER LIGNE MENU, hormis le fait qu'elle permet d'insérer des commandes de menu partout dans le menu alors que AJOUTER LIGNE MENU les ajoute toujours à la fin du menu.

Reportez-vous à la description de la commande AJOUTER LIGNE MENU pour plus de détails sur la définition des commandes de menus passée dans libelléLigne et sur l'action du paramètre *.

Le paramètre facultatif sousMenu vous permet de désigner un menu comme ligne insérée et donc de définir un sous-menu hiérarchique. Vous devez passer dans ce paramètre une référence de menu (chaîne de type *RefMenu*) désignant un menu créé par exemple à l'aide de la commande Creer menu. Si la commande insère plusieurs lignes de menus, le sous-menu est associé à la première ligne.

Important : Les nouvelles lignes n'ont pas de méthodes ou d'actions associées. Vous devez leur associer une action ou une méthode via les commandes FIXER PROPRIETE LIGNE MENU ou FIXER METHODE LIGNE MENU ou encore les gérer à partir d'une méthode formulaire qui utilise la fonction Menu choisi.

Exemple

L'exemple suivant crée un menu constitué de deux commandes auxquelles il affecte une méthode :

```
refMenu:=Creer menu
AJOUTER LIGNE MENU(refMenu;"Caractères")
FIXER METHODE LIGNE MENU(refMenu;1;"GestCaracDial")
INSERER LIGNE MENU(refMenu;1;"Paragraphes")
FIXER METHODE LIGNE MENU(refMenu;2;"GestParDial")
```

Référence

AJOUTER LIGNE MENU, FIXER PARAMETRE LIGNE MENU.

LIRE ICONE LIGNE MENU (menu; ligneMenu; reflcône{; process})

Paramètre	Type		Description
menu	RefMenu | Num	→	Référence de menu ou Numéro de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
reflcône	Var Texte | Var E. long	←	Nom ou numéro de l'image associée à la ligne de menu
process	Numérique	→	Numéro de process

Description

La commande LIRE ICONE LIGNE MENU retourne dans la variable reflcône la référence de l'icône éventuellement associée à la ligne de menu désignée par les paramètres menu et ligneMenu. Cette référence est le nom ou le numéro de l'image.

L'icône associée à une ligne de menu est ajoutée dans la barre d'outils de l'application.

Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu) ou un numéro de menu. Si vous passez un identifiant unique, le paramètre process est inutile et sera ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

Si l'icône avait été définie à l'aide d'une image de la bibliothèque, la commande retourne dans reflcône le nom ou le numéro de l'image en fonction du type de la variable passée dans ce paramètre. Si l'icône avait été définie à l'aide d'une image stockée dans le dossier **Resources** de la base, la commande retourne dans reflcône le chemin d'accès de l'image. Si vous n'attribuez pas de type spécifique à la variable reflcône, par défaut le nom de l'image est retourné (type texte).

Si aucune icône n'est associée à la ligne, la commande retourne une valeur vide.

Référence

FIXER ICONE LIGNE MENU.

LIRE LIGNES MENU (menu; tabTitresMenus; tabRefsMenus)

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ou Numéro de menu
tabTitresMenus	Tab Chaîne (32)	←	Tableau des libellés du menu
tabRefsMenus	Tab Chaîne (16)	←	Tableau des références du menu

Description

La commande LIRE LIGNES MENU retourne dans les tableaux tabTitresMenu et tabRefsMenu les libellés et les identifiants de toutes les lignes du menu ou de la barre de menus désigné(e) par le paramètre menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu), un numéro de barre de menus ou une référence de barre de menus obtenue via la commande Lire reference barre menu.

Lorsqu'aucune référence de menu n'est rattachée à une ligne, une chaîne vide est retournée dans l'élément de tableau correspondant.

Exemple

Vous souhaitez connaître le contenu de la barre de menus du process courant :

```
TABLEAU TEXTE(tabTitresMenu;0)
TABLEAU TEXTE(tabRefsMenu;0)
RefBarreMenu:=Lire reference barre menu(Process de premier plan)
LIRE LIGNES MENU(RefBarreMenu;tabTitresMenu;tabRefsMenu)
```

Lire marque ligne menu

version 11 (Modifiée)

Lire marque ligne menu (menu; ligneMenu{; process}) → Alpha

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence de process
Résultat	Alpha	←	Marque de ligne de menu courante

Description

La commande Lire marque ligne menu retourne la marque (ou "coche") de la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous omettez le paramètre process, Lire marque ligne menu s'applique à la barre de menus du process courant. Sinon, Lire marque ligne menu s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Si la ligne de menu n'a pas de marque ou si le paramètre ligneMenu désigne un sous-menu hiérarchique, Lire marque ligne menu retourne une chaîne vide.

Note : Pour plus d'informations sur les marques des lignes de menus sous Mac OS et Windows, reportez-vous à la description de la commande FIXER MARQUE LIGNE MENU.

Exemple

L'exemple suivant inverse l'état marqué d'une ligne de menu :

```
FIXER MARQUE LIGNE MENU($vlMenu;$vlItem;Caractere(18)*
                        Num(Lire marque ligne menu($vlMenu;$vlItem)=""))
```

Référence

FIXER MARQUE LIGNE MENU.

Lire methode ligne menu

Lire methode ligne menu (menu; ligneMenu{; process}) → Chaîne

Paramètre	Type		Description
menu	RefMenu | Num	→	Référence de menu ou Numéro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
process	Entier long	→	Numéro de process
Résultat	Chaîne	←	Nom de la méthode

Description

La commande Lire methode ligne menu retourne le nom de la méthode projet 4D associée à la ligne de menu désignée par les paramètres menu et ligneMenu.
Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu) ou un numéro de menu. Si vous passez un identifiant unique, le paramètre process est inutile et sera ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

La commande retourne le nom de la méthode 4D sous la forme d'une chaîne de caractères (expression). Si aucune méthode n'est associée à la ligne de menu, la commande retourne une chaîne vide.

Référence

FIXER METHODE LIGNE MENU.

Lire modificateurs ligne menu (menu; ligneMenu{; process}) → Nombre

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ouNuméro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
process	Entier long	→	Numéro de process
Résultat	Nombre	←	Touche(s) de modification associée(s) à la ligne de menu

Description

La commande Lire modificateurs ligne menu retourne le ou les modificateur(s) additionnel(s) associé(s) au raccourci standard de la ligne de menu désignée par les paramètres menu et ligneMenu.
Le raccourci standard est composé de la touche **Commande** (Mac OS) ou **Ctrl** (Windows) et d'une touche personnalisée. Le raccourci standard est géré via les commandes FIXER RACCOURCI LIGNE MENU et Lire touche ligne menu.

Les modificateurs additionnels sont la touche **Majuscule** et la touche **Option** (Mac OS) / **Alt** (Windows). Ces modificateurs ne sont utilisables que si un raccourci standard a été défini au préalable.

La valeur numérique retournée par la commande correspond au code de la ou des touche(s) de modification additionnelles. Les codes des touches sont les suivants :
• **Majuscule** = 512
• **Option** (Mac OS) ou **Alt** (Windows) = 2048
Si les deux touches sont utilisées, leur valeur est cumulée.

Note : Vous pouvez évaluer la valeur retournée à l'aide des constantes Masque touche majuscule et Masque touche option du thème "Evénements (Modifiers)".

Si la ligne de menu n'a pas de touche de modification associée, la commande retourne 0.
Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu) ou un numéro de menu.
Si vous passez un identifiant unique, le paramètre process est inutile et sera ignoré s'il est passé.

Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

Exemple

Reportez-vous à l'exemple de la commande Lire touche ligne menu.

Référence

FIXER RACCOURCI LIGNE MENU, Lire touche ligne menu.

LIRE PROPRIETE LIGNE MENU (menu; ligneMenu; propriété; valeur{; process})

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ou Numéro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
propriété	Chaîne	→	Type de propriété
valeur	Expression	←	Valeur de la propriété
process	Entier long	→	Numéro de process

Description

La commande LIRE PROPRIETE LIGNE MENU retourne dans le paramètre valeur la valeur courante de la propriété de la ligne de menu désignée par les paramètres menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Vous pouvez passer dans menu un identifiant unique de menu (RefMenu) ou un numéro de menu. Si vous passez un identifiant unique, le paramètre process est inutile et sera ignoré s'il est passé. Si vous passez un numéro de menu, la commande prendra en compte le menu correspondant dans la barre de menus principale du process courant. Si vous souhaitez désigner un autre process, passez son numéro dans le paramètre facultatif process.

Passez dans le paramètre propriété la propriété dont vous souhaitez obtenir la valeur. Vous pouvez utiliser l'une des constantes du thème "Propriétés des lignes de menu" ou une chaîne correspondant à une propriété personnalisée. Pour plus d'informations sur les propriétés des menus et leurs valeurs, reportez-vous à la description de la commande FIXER PROPRIETE LIGNE MENU.

Référence

FIXER PROPRIETE LIGNE MENU.

Lire reference barre menu

Lire reference barre menu {(process)} → RefMenu

Paramètre	Type		Description
process	Numérique	→	Numéro de référence du process
Résultat	RefMenu	←	Identifiant de la barre de menus

Description

La commande Lire reference barre menu renvoie l'identifiant unique de la barre de menus courante ou de la barre de menus d'un process spécifique.

Si la barre de menus a été créée par la commande Creer menu, cet identifiant correspond à la référence unique du menu créé. Sinon, la commande retourne un identifiant interne spécifique. Dans tous les cas, cet identifiant RefMenu pourra être utilisé pour référencer la barre de menus par toutes les autres commandes du thème.

Le paramètre process permet de désigner le process duquel vous souhaitez obtenir l'identifiant de la barre de menus courante. Si vous omettez ce paramètre, la commande retourne l'identifiant de la barre de menus du process courant.

Exemple

Reportez-vous à l'exemple de la commande LIRE LIGNES MENU.

Référence

FIXER BARRE MENUS.

Lire parametre ligne menu

version 11.4 (Modifiée)

Lire parametre ligne menu (menu; ligneMenu) → Chaîne

Paramètre	Type		Description
menu	RefMenu | Entier long	→	Référence de menu ou Numéro de menu
ligneMenu	Entier long	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée au menu
Résultat	Chaîne	←	Paramètre personnalisé de la ligne de menu

Description

La commande Lire parametre ligne menu retourne la chaîne de caractères personnalisée associée à la ligne de menu désignée par les paramètres menu et ligneMenu. Cette chaîne doit avoir été préalablement définie à l'aide de la commande FIXER PARAMETRE LIGNE MENU.

Référence

FIXER PARAMETRE LIGNE MENU, Lire parametre ligne menu selectionnee, Pop up menu dynamique.

Lire parametre ligne menu selectionnee

version 11.4 (Modifiée)

Lire parametre ligne menu selectionnee → Chaîne

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Chaîne	←	Paramètre personnalisé de la ligne de menu

Description

La commande Lire parametre ligne menu selectionnee retourne la chaîne de caractères personnalisée associée à la ligne de menu sélectionnée. Ce paramètre doit avoir été préalablement défini à l'aide de la commande FIXER PARAMETRE LIGNE MENU.
Si aucune ligne de menu n'a été sélectionnée, la commande retourne une chaîne vide "".

Référence

FIXER PARAMETRE LIGNE MENU, Lire parametre ligne menu, Pop up menu dynamique.

Lire style ligne menu (menu; ligneMenu{; process}) → Numérique

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence de process
Résultat	Numérique	←	Style courant de la ligne de menu

Description

La commande Lire style ligne menu retourne le style de police de la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous omettez le paramètre process, Lire style ligne menu s'applique à la barre de menus du process courant. Sinon, Lire style ligne menu s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Lire style ligne menu retourne une combinaison (une ou une somme) des constantes prédéfinies suivantes, placées dans le thème Styles de caractères :

Constante	Type	Valeur
Standard	Entier long	0
Gras	Entier long	1
Italique	Entier long	2
Souligné	Entier long	4

Exemple

Si, par exemple, vous voulez tester si une ligne de menu est affichée en gras, vous écrivez :

```
Si ((Lire style ligne menu($vlMenu;$vlItem) & Gras)#0)
   `...
Fin de si
```

Référence

FIXER STYLE LIGNE MENU.

Lire texte ligne menu

version 11 (Modifiée)

Lire texte ligne menu (menu; ligneMenu{; process}) → Alpha

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence de process
Résultat	Alpha	←	Libellé de la ligne de menu

Description

La commande Lire texte ligne menu retourne le libellé de la commande de menu dont le numéro ou la référence de menu et le numéro de commande ont été passés dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous ne passez pas le paramètre process, Lire texte ligne menu est appliquée à la barre de menus du process courant. Sinon Lire texte ligne menu est appliquée à la barre de menus du process dont la référence est passée dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Référence

FIXER TEXTE LIGNE MENU, Lire touche ligne menu.

Lire titre menu

version 11 (Modifiée)

Lire titre menu (menu{; process}) → Alpha

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
process	Numérique	→	Numéro de référence de process
Résultat	Alpha	←	Titre du menu

Description

La commande Lire titre menu retourne le titre du menu dont vous avez passé le numéro ou la référence dans menu.

Si vous omettez le paramètre process, Lire titre menu s'applique à la barre de menus du process courant. Sinon, Lire titre menu s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Référence

Nombre de menus.

Lire touche ligne menu

version 11 (Modifiée)

Lire touche ligne menu (menu; ligneMenu{; process}) → Numérique

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de la ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence de process
Résultat	Numérique	←	Code de caractère de de la touche de raccourci standard associée à la ligne de menu

Description

La commande Lire touche ligne menu retourne le code de la touche **Ctrl** (sous Windows) ou **Commande** (Mac OS) utilisée comme raccourci clavier pour la commande de menu dont le numéro ou la référence de menu et le numéro de ligne ont été passés dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si vous ne passez pas le paramètre process, Lire touche ligne menu est appliquée à la barre de menus du process courant. Sinon, Lire touche ligne menu est appliquée à la barre de menus du process dont la référence est passée dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Si la ligne de menu n'a pas de touche de raccourci associée ou si le paramètre ligneMenu désigne un sous-menu hiérarchique, Lire touche ligne menu retourne 0 (zéro).

Exemple

Pour obtenir le raccourci clavier associé à une ligne de menu, il est utile de mettre en place une structure de programmation du type suivant :

```
Si(Lire touche ligne menu(monmenu;1) # 0)
    $modifiers:=Lire modificateurs ligne menu(monmenu;1)
    Au cas ou
        : ($modifiers=Masque touche option)
            ...
        : ($modifiers=Masque touche majuscule)
            ...
        : ($modifiers=Masque touche option + Masque touche majuscule)
            ...
    Fin de cas
Fin de si
```

Référence

FIXER RACCOURCI LIGNE MENU, Lire touche ligne menu.

Menu choisi

version 11 (Modifiée)

Menu choisi {(sousMenu)} → Numérique

Paramètre	Type		Description
sousMenu	RefMenu	←	Référence du menu contenant la ligne sélectionnée
Résultat	Numérique	←	Commande de menu sélectionnée Mot machine haut = n° de menu Mot machine bas = n° de commande de menu

Description

Menu choisi ne s'utilise que lorsqu'un formulaire est affiché. Cette fonction détecte la commande de menu choisie dans un menu et, dans le cas d'un sous-menu hiérarchique, retourne la référence du sous-menu.

Astuce : A chaque fois que cela est possible, utilisez des méthodes associées à des commandes de menus dans une barre associée (avec un numéro de barre négatif) plutôt que d'appeler Menu choisi. Les barres de menus associées sont plus faciles à gérer, puisqu'il n'est pas nécessaire de tester leur sélection.

La commande Menu choisi permet de travailler avec des sous-menus hiérarchiques. En cas de sélection d'une ligne d'un menu hiérarchique au-delà du premier niveau, la commande retourne dans le paramètre facultatif sousMenu la référence (type *RefMenu*, chaîne de 16 caractères) du sous-menu auquel appartient la ligne sélectionnée. Si la commande du menu ne contient pas de sous-menu hiérarchique, ce paramètre reçoit une chaîne vide.

Menu choisi retourne le numéro système du menu sélectionné sous forme d'Entier long. Pour obtenir le numéro du menu, divisez Menu choisi par 65536 et convertissez le résultat en Entier. Pour obtenir le numéro de la commande de menu, calculez le modulo de Menu choisi avec le coefficient 65536. Utilisez les formules suivantes pour calculer le numéro du menu et de la commande de menu :

```
Menu := Menu choisi \ 65536
Ligne de menu := Menu choisi % 65536
```

Vous pouvez également extraire ces valeurs à l'aide des Opérateurs sur les bits, comme dans l'exemple suivant :

```
Menu := (Menu choisi & 0xFFFF0000) >> 16
Ligne de menu := Menu choisi & 0xFFFF
```

Menu choisi retourne 0 si aucune commande de menu n'est sélectionnée.

Exemple

La méthode formulaire suivante utilise la fonction Menu choisi pour fournir les arguments "menu" et "ligne de menu" à FIXER MARQUE LIGNE MENU :

```
Au cas ou
: (Evenement formulaire=Sur menu sélectionné)
   C_ALPHA(16;$refMenuIncludingItem)
   C_ENTIER LONG($ref;$NumMenu;$NumMenuItem)
   $ref:= Menu choisi ($refMenuIncludingItem)
   $NumMenu:=$ref\65536
   $NumMenuItem:=$ref%65536
   FIXER MARQUE LIGNE MENU ($refMenuIncludingItem; $NumMenuItem;
                                                    Caractere(18))
Fin de cas
```

Note : L'événement Sur menu selectionné n'est pas activé si aucune ligne n'est sélectionnée, $refmenuincludingItem est toujours renseigné et $NumMenu vaut 0 si le menu n'est pas un des menus de la barre.

Référence

Gestion des menus.

Nombre de lignes de menu

version 11 (Modifiée)

Nombre de lignes de menu (menu{; process}) → Numérique

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
process	Numérique	→	Numéro de référence de process
Résultat	Numérique	←	Nombre de lignes du menu

Description

La commande Nombre de lignes de menu retourne le nombre de lignes (commandes) de menus présentes dans le menu dont vous avez passé le numéro ou la référence dans menu.

Si vous omettez le paramètre process, Nombre de lignes de menu s'applique à la barre de menus du process courant. Sinon, Nombre de lignes de menu s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Référence

Nombre de menus.

Nombre de menus

Nombre de menus {(process)} → Numérique

Paramètre	Type		Description
process	Numérique	→	Numéro de référence de process
Résultat	Numérique	←	Nombre de menus de la barre de menus courante

Description

La commande Nombre de menus retourne le nombre de menus présents dans la barre de menus.

Si vous omettez le paramètre process, Nombre de menus s'applique à la barre de menus du process courant. Sinon, Nombre de menus s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Référence

Nombre de lignes de menu.

Pop up menu dynamique

Pop up menu dynamique (menu{; parDéfaut{; coordX{; coordY}}}) → Chaîne

Paramètre	Type		Description
menu	RefMenu	→	Référence de menu
parDéfaut	Chaîne	→	Paramètre de l'élément sélectionné par défaut
coordX	Numérique	→	Coordonnée X du coin supérieur gauche
coordY	Numérique	→	Coordonnée Y du coin supérieur gauche
Résultat	Chaîne	←	Paramètre de l'élément de menu sélectionné

Description

La commande Pop up menu dynamique fait apparaître un pop up menu hiérarchique à l'emplacement courant de la souris ou à l'emplacement défini par les paramètres facultatifs coordX et coordY.
Le menu hiérarchique utilisé doit avoir été créé à l'aide de la commande Creer menu. La référence retournée par Creer menu doit être passée dans le paramètre menu.

Conformément aux règles standard d'interface, cette commande doit généralement être appelée en réponse à un clic droit, ou lorsque le bouton reste enfoncé un certain laps de temps (menu contextuel par exemple).

Le paramètre facultatif parDéfaut vous permet de définir un élément du pop up menu sélectionné par défaut lorsque celui-ci apparaît. Passez dans ce paramètre la chaîne personnalisée associée à l'élément de menu. Cette chaîne doit avoir été préalablement définie à l'aide de la commande FIXER PARAMETRE LIGNE MENU. Si vous ne passez pas ce paramètre, le premier élément du menu sera sélectionné par défaut.

Les paramètres facultatifs coordX et coordY permettent de désigner l'emplacement du pop up menu à afficher. Passez respectivement dans coordX et coordY les coordonnées horizontale et verticale du coin supérieur gauche du menu. Ces coordonnées doivent être exprimées en pixels dans le système de coordonnées local au formulaire courant. Ces deux paramètres doivent être passés ensemble ; si un seul est passé, il est ignoré.

Si vous souhaitez afficher un pop up associé à un bouton 3D, il suffit de ne pas passer les paramètres facultatifs coordX et coordY. Dans ce cas, 4D calcule automatiquement l'emplacement du menu par rapport au bouton en fonction des normes d'interface de la plate-forme courante.

Si une ligne de menu a été sélectionnée, la commande retourne sa chaîne de caractères personnalisée associée (telle que définie à l'aide de la commande FIXER PARAMETRE LIGNE MENU). Sinon, la commande retourne une chaîne vide.

Note : La commande Pop up menu (thème "Interface utilisateur") permet de créer des pop up menus basés sur du texte.

Référence

FIXER PARAMETRE LIGNE MENU, Lire parametre ligne menu, Lire parametre ligne menu selectionnee, Pop up menu.

SUPPRIMER LIGNE MENU

version 11 (Modifiée)

SUPPRIMER LIGNE MENU (menu; ligneMenu{; process})

Paramètre	Type		Description
menu	Num | RefMenu	→	Numéro de menu ou Référence de menu
ligneMenu	Numérique	→	Numéro de ligne de menu ou -1 pour la dernière ligne ajoutée
process	Numérique	→	Numéro de référence de process

Description

La commande SUPPRIMER LIGNE MENU supprime la ligne de menu dont vous avez passé le numéro ou la référence de menu et le numéro de ligne dans menu et ligneMenu. Vous pouvez passer -1 dans ligneMenu afin de désigner la dernière ligne ajoutée au menu.

Si la ligne de menu désignée par menu et ligneMenu est elle-même un menu géré par référence et créé par exemple à l'aide la commande Creer menu, SUPPRIMER LIGNE MENU supprimera uniquement l'instance de ligneMenu dans menu. Le sous-menu référencé par ligneMenu continuera d'exister en mémoire. Vous devez utiliser la commande EFFACER MENU afin de supprimer définitevement un menu géré par référence.
Cette commande fonctionne également avec une barre de menus créée avec la commande Creer menu et installée avec la commande FIXER BARRE MENUS.

Si vous omettez le paramètre process, SUPPRIMER LIGNE MENU s'applique à la barre de menus du process courant. Sinon, SUPPRIMER LIGNE MENU s'applique à la barre de menus du process dont vous avez passé le numéro dans process.

Note : Si vous passez un paramètre RefMenu dans menu, le paramètre process est inutile et sera ignoré.

Note : Pour soigner l'ergonomie de votre interface, ne laissez pas accessible un menu ne comportant aucune ligne.

Référence

AJOUTER LIGNE MENU, INSERER LIGNE MENU.

39

Messages

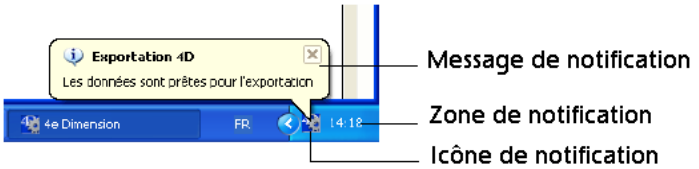
AFFICHER NOTIFICATION (titre; contenu{; délai})

Paramètre	Type		Description
titre	Alpha 255	→	Titre de la notification
contenu	Alpha 255	→	Texte de la notification
délai	Numérique	→	Délai d'affichage en secondes

Note : Cette commande fonctionne sous Windows uniquement.

Description

La commande AFFICHER NOTIFICATION provoque l'affichage d'un message dans la zone de notification de la barre des tâches de Windows :



Ce type de message est généralement utilisé par le système ou les applications pour informer l'utilisateur d'un événement (déconnexion réseau, disponibilité de mises à jour, etc.)

Passez dans les paramètres titre et contenu le titre et le texte du message à afficher (dans l'exemple ci-dessus, le titre est "Exportation 4D"). Vous pouvez saisir jusqu'à 255 caractères.

Par défaut, la fenêtre du message reste affichée jusqu'à ce que l'utilisateur clique sur sa case de fermeture. Si vous passez le paramètre facultatif délai, la fenêtre sera automatiquement refermée à l'issue de la durée définie si l'utilisateur n'a pas cliqué sur la case de fermeture. A noter que l'icône de notification restera affichée jusqu'à la fin du délai, même si l'utilisateur a refermé le message.

Référence

ALERTE.

ALERTE (message{; libelléBoutonOK})

Paramètre	Type		Description
message	Alpha	→	Message à afficher dans la boîte de dialogue d'alerte
libelléBoutonOK	Alpha	→	Libellé du bouton OK

Description

La commande ALERTE affiche une boîte de dialogue d'alerte composée d'une icône, d'un message et d'un bouton OK.

Vous passez le message à afficher dans le paramètre message. Ce message peut contenir jusqu'à 255 caractères. Si la longueur ou la largeur des caractères est trop importante par rapport à la zone du message, il sera tronqué.

Par défaut, le libellé du bouton OK est “OK”. Si vous voulez changer ce libellé, passez le nouveau libellé dans le paramètre optionnel libelléBoutonOK. Si nécessaire, la largeur du bouton OK est augmentée vers la gauche pour contenir ce nouveau libellé.

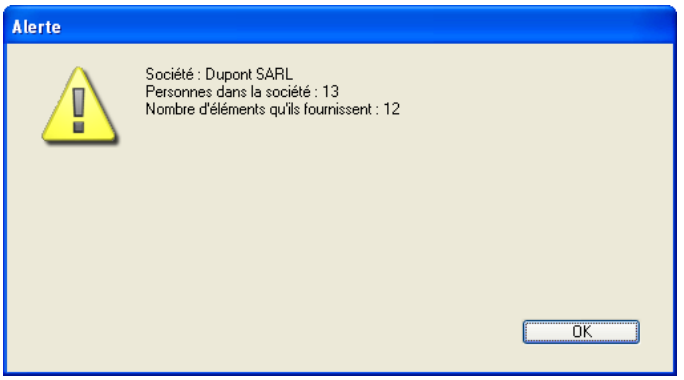
Note : N'appelez pas la commande ALERTE dans une méthode formulaire ou une méthode objet qui gère l'événement formulaire Sur activation ou Sur désactivation, car cela provoquerait une boucle sans fin.

Exemples

(1) L'exemple suivant appelle une boîte de dialogue d'alerte qui affiche des informations sur une société. Notez que le message contient des retours chariot (Caractere(13)) qui forcent le texte à passer sur la ligne suivante :

```
ALERTE("Société : "+[Sociétés]Nom+Caractere(13)+"Personnes dans la société : "+
    Chaine(Enregistrements trouves([Personne]))+Caractere(13)+"Nombre d'éléments
        qu'ils fournissent : "+Chaine (Enregistrements trouves([Eléments])))
```

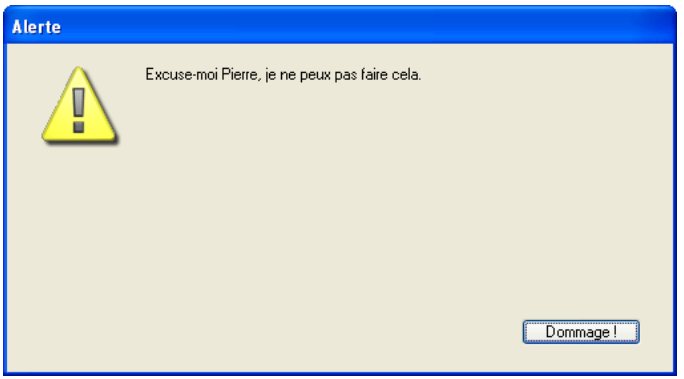
Voici la boîte de dialogue d'alerte affichée (sous Windows) par notre exemple :



(2) Voici un autre exemple :

ALERTE("Excuse-moi Pierre, je ne peux pas faire cela.";"Dommage !")

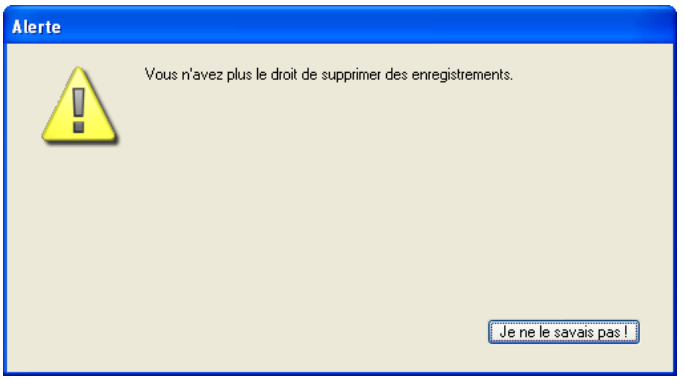
Cette instruction affichera (sous Windows) la boîte de dialogue d'alerte suivante :



(3) Voici un autre exemple :

ALERTE("Vous n'avez plus le droit de supprimer des enregistrements.";"Je ne le savais pas !")

Ce code affiche la boîte de dialogue d'alerte suivante :



Référence

AFFICHER NOTIFICATION, CONFIRMER, Demander.

CONFIRMER (message{; libelléBoutonOK{; libelléBoutonAnn}})

Paramètre	Type		Description
message	Alpha	→	Message à afficher dans la boîte de dialogue de confirmation
libelléBoutonOK	Alpha	→	Libellé du bouton OK
libelléBoutonAnn	Alpha	→	Libellé du bouton Annuler

Description

La commande CONFIRMER affiche une boîte de dialogue de confirmation qui se compose d'une icône, d'un message, d'un bouton OK et d'un bouton Annuler.

Les boîtes de dialogue de confirmation ou d'alerte sont utilisées pour afficher des informations (comme des messages d'erreur) qui ne nécessitent pas d'informations en retour.

Vous passez le message à afficher dans le paramètre message. Ce message peut contenir jusqu'à 255 caractères. Si la longueur ou largeur des caractères est trop importante par rapport à la zone d'affichage, le message sera tronqué.

Par défaut, le libellé du bouton OK est “OK” et le libellé du bouton Annuler est “Annuler”. Si vous voulez modifier le libellé de ces boutons, passez le nouveau libellé dans les paramètres optionnels libelléBoutonOK et libelléBouton Ann. Si nécessaire, les boutons sont agrandis vers la gauche en fonction de la taille des libellés que vous avez saisis.

Le bouton OK est le bouton par défaut. L'utilisateur peut cliquer sur le bouton OK ou appuyer sur la touche **Entrée** pour valider la boîte de dialogue, la variable système OK prend alors la valeur 1. L'utilisateur peut cliquer sur le bouton Annuler pour annuler la boîte de dialogue, la variable système OK prend alors la valeur 0.

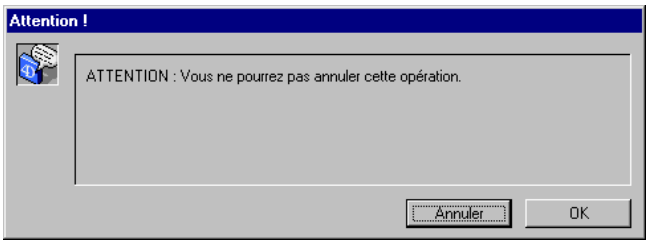
Conseil : N'appelez pas la commande CONFIRMER dans une méthode formulaire ou objet qui gère l'événement Sur activation ou Sur désactivation, car cela provoquerait une boucle sans fin.

Exemples

(1) L'exemple ci-dessous :

```
CONFIRMER("ATTENTION : Vous ne pourrez pas annuler cette opération.")
Si (OK=1)
    TOUT SELECTIONNER([Vieilles choses])
    SUPPRIMER SELECTION([Vieilles choses])
Sinon
    ALERTE ("Opération annulée.")
Fin de si
```

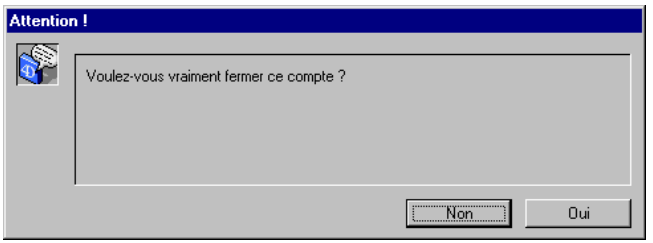
... provoquera l'affichage de la boîte de dialogue de confirmation suivante (sous Windows) :



(2) La ligne :

```
CONFIRMER("Voulez-vous vraiment fermer ce compte ?";"Oui";"Non")
```

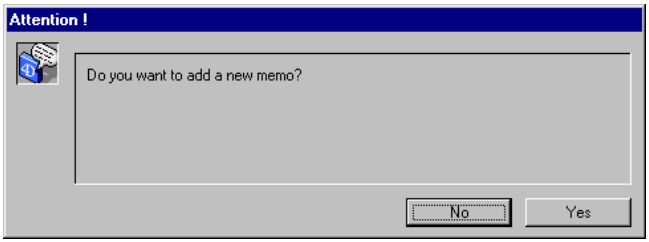
... provoquera l'affichage de la boîte de dialogue de confirmation suivante (sous Windows) :



(3) Vous développez une application 4D pour le marché international. Vous avez écrit une méthode projet qui retourne du texte traduit à partir d'une version française. Vous avez également rempli un tableau nommé asLocalizedUIMessages dans lequel vous stockez les mots les plus courants. Dans ce cas, la ligne :

CONFIRMER(*INTL Text* ("Voulez-vous ajouter un nouveau mémo ?");
asLocalizedUIMessages{kLoc_OUI};asLocalizedUIMessages{kLoc_NON})

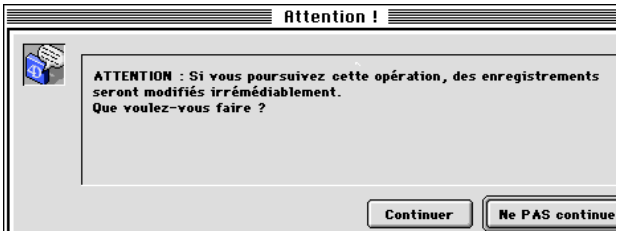
... pourrait afficher la boîte de dialogue de confirmation (sous Windows) suivante :



(4) La ligne :

CONFIRMER("ATTENTION : Si vous poursuivez cette opération, des enregistrements
seront "+"modifiés irrémédiablement."+**Caractere**(13)+
"Que voulez-vous faire ?";"Ne PAS continuer";"Continuer")

... provoque l'affichage de la boîte de dialogue de confirmation suivante (sous Mac OS) :



Référence

ALERTE, Demander.

Demander (message{; réponseDéfaut{; titreBoutonOK{; titreBoutonAnn}}}) → Alpha

Paramètre	Type		Description
message	Alpha	→	Message à afficher dans la boîte de dialogue
réponseDéfaut	Alpha	→	Valeur par défaut dans la zone de saisie de texte
titreBoutonOK	Alpha	→	Libellé du bouton OK
titreBoutonAnn	Alpha	→	Libellé du bouton Annuler
Résultat	Alpha	←	Valeur saisie par l'utilisateur

Description

La fonction Demander affiche une boîte de dialogue de demande d'informations composée d'un message, d'une zone de saisie de texte, d'un bouton **OK** et d'un bouton **Annuler**.

Vous passez le message à afficher dans le paramètre message. Ce message peut contenir jusqu'à 255 caractères. Il peut cependant apparaître tronqué, en fonction de sa taille et de la largeur des caractères, s'il est supérieur à la capacité d'affichage de la zone de message.

Par défaut, le libellé du bouton OK est “OK” et celui du bouton Annuler est “Annuler”. Si vous voulez modifier ces libellés, passez d'autres valeurs dans les paramètres optionnels titreBoutonOK et titreBoutonAnn. Si nécessaire, les boutons sont agrandis vers la gauche, en fonction de la taille des libellés que vous avez saisis.

Le bouton OK est le bouton par défaut. L'utilisateur peut cliquer sur le bouton OK ou appuyer sur la touche Entrée pour valider la boîte de dialogue, mettant ainsi la variable système OK à 1. Il peut également cliquer sur le bouton Annuler pour annuler la boîte de dialogue, mettant ainsi la variable système OK à 0.

L'utilisateur peut taper des caractères dans la zone de saisie de texte. Pour définir une valeur par défaut, passez le texte par défaut dans le paramètre réponseDéfaut. Si l'utilisateur clique sur le bouton OK, Demander retourne le texte. Si l'utilisateur clique sur le bouton Annuler, Demander retourne une chaîne vide (""). Si la réponse doit être une valeur numérique ou une date, convertissez la chaîne retournée par Demander dans le type souhaité à l'aide des fonctions Num et Date.

Note : N'appelez pas la fonction Demander dans une méthode formulaire ou objet qui gère l'événement Sur activation ou Sur desactivation car cela provoquerait une boucle sans fin.

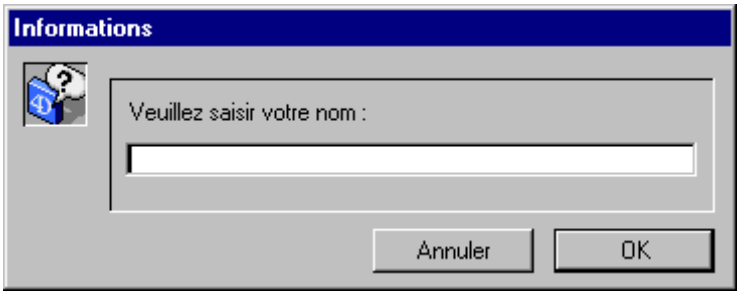
Conseil : Si vous voulez récupérer plusieurs informations de l'utilisateur, construisez un formulaire approprié et appelez-le avec la commande DIALOGUE, plutôt que d'afficher une succession de boîtes de dialogue du type Demander.

Exemples

(1) La ligne de code :

```
$vsAffiche := Demander ("Veuillez saisir votre nom :")
```

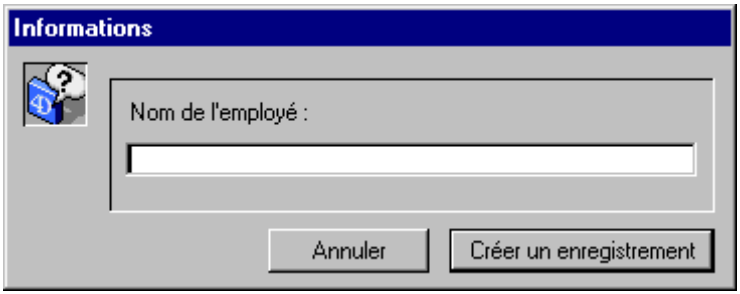
... provoquera l'affichage de la boîte de dialogue suivante :



(2) Le code suivant :

```
$vsAffiche:= Demander ("Nom de l'employé :";"";"Créer un enregistrement";"Annuler")
Si (OK=1)
    AJOUTER ENREGISTREMENT([Employés])
        ` Note : $vsAffiche est alors copiée dans le champ [Employés]Nom
        ` lors de l'événement formulaire Sur chargement de la méthode formulaire
Fin de si
```

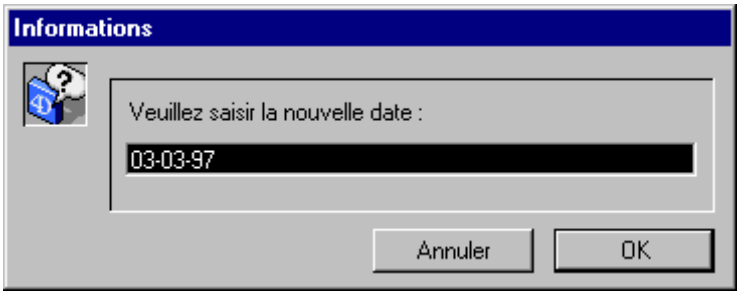
... provoquera l'affichage de la boîte de dialogue suivante :



(3) La ligne de code :

$vdAffiche := **Date** (**Demander** ("Veuillez saisir la nouvelle date :";**Chaine** (**Date du jour**)))

... provoquera l'affichage de la boîte de dialogue suivante :



Référence

ALERTE, CONFIRMER.

LAISSER MESSAGES

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Reportez-vous à la description de la commande SUPPRIMER MESSAGES.

MESSAGE (message)

Paramètre	Type		Description
message	Alpha	→	Message à afficher

Description

La commande MESSAGE affiche message à l'écran dans une fenêtre spéciale de message qui est ouverte et refermée à chaque fois que vous l'appelez (à moins que vous ne travailliez dans une fenêtre préalablement ouverte par la commande Creer fenetre, cf. ci-dessous). Le message est temporaire et est effacé dès qu'un formulaire est affiché ou dès que l'exécution de la méthode est stoppée. Si une autre commande MESSAGE est exécutée, le précédent message est effacé.

MESSAGE est généralement utilisée pour informer l'utilisateur du déroulement d'une action.

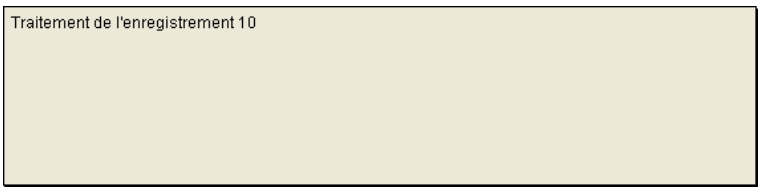
Si une fenêtre a été ouverte par la commande Creer fenetre, tous les appels ultérieurs à la commande MESSAGE affichent les messages dans cette fenêtre. Cette fenêtre se comporte en quelque sorte comme un terminal :
• Chaque message successif n'efface pas le précédent, les messages se placent les uns à la suite des autres.
• Si un message est plus large que la fenêtre, 4D insère automatiquement un retour à la ligne.
• Si le message contient plus de lignes que ne peut en afficher la fenêtre, 4D fait automatiquement défiler le message dans la fenêtre.
• Si vous souhaitez contrôler les retours à la ligne, insérez vos propres retours chariot dans votre texte, à l'aide de Caractere(13).
• Vous pouvez appeler la commande POSITION MESSAGE pour afficher le texte à un emplacement particulier dans la fenêtre.
• Vous pouvez appeler la commande EFFACER FENETRE pour effacer le contenu de la fenêtre.
• La fenêtre est une fenêtre d'affichage statique : son contenu n'est pas redessiné lorsque d'autres fenêtres s'affichent par-dessus.

Exemples

(1) L'exemple suivant traite une sélection d'enregistrements et appelle la commande MESSAGE pour informer l'utilisateur de la progression de l'opération :

```
Boucle($vlEnregistrement;1;Enregistrements trouves([touteTable]))
    MESSAGE ("Traitement de l'enregistrement "+Chaine($vlEnregistrement))
        ` Faire quelque chose avec l'enregistrement
    ENREGISTREMENT SUIVANT([touteTable])
Fin de boucle
```

La fenêtre suivante s'affiche puis disparaît à chaque appel de MESSAGE :



(2) Afin d'éliminer le "clignotement" de la fenêtre, il est préférable, comme dans ce deuxième exemple, d'afficher les messages dans une fenêtre ouverte par l'intermédiaire de la commande Creer fenetre :

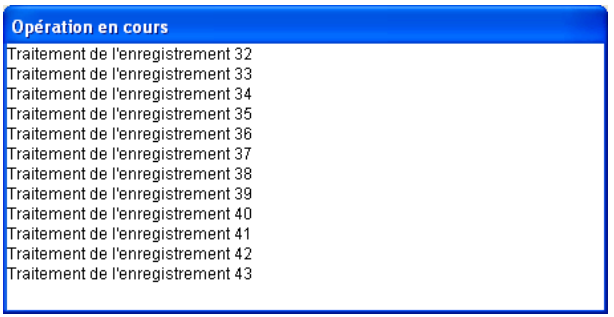
```
Creer fenetre(50;50;500;250;5;"Opération en cours")
Boucle($vlEnregistrement;1;Enregistrements trouves([touteTable]))
    MESSAGE ("Traitement de l'enregistrement "+Chaine($vlEnregistrement))
        ` Faire quelque chose avec l'enregistrement
    ENREGISTREMENT SUIVANT([touteTable])
Fin de boucle
FERMER FENETRE
```

Le résultat est le suivant (sous Windows) :

(3) En ajoutant un retour chariot, vous améliorez la présentation :

```
Creer fenetre(50;50;500;250;5;"Opération en cours")
Boucle($vlEnregistrement;1;Enregistrements trouves([touteTable]))
   MESSAGE ("Traitement de l'enregistrement "+Chaine($vlEnregistrement)+
                                                  Caractere(Retour chariot))
      ` Faire quelque chose avec l'enregistrement
   ENREGISTREMENT SUIVANT([touteTable])
Fin de boucle
FERMER FENETRE
```

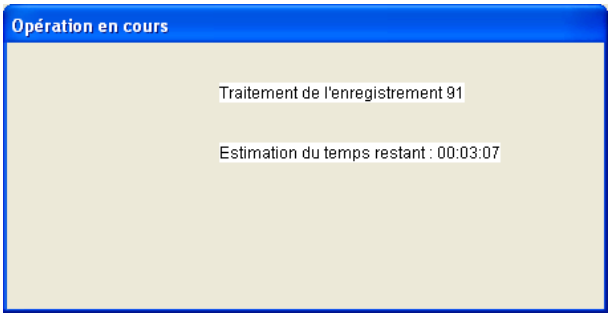
Voici le résultat (Sous Windows) :



(4) A l'aide de la commande POSITION MESSAGE et de l'écriture de quelques lignes supplémentaires, la présentation s'améliore nettement :

```
Creer fenetre(50;50;500;250;5;"Opération en cours")
$vlNbEnregistrements:=Enregistrements trouves([touteTable])
$vhHeureDébut:=Heure courante
Boucle($vlEnregistrement;1;$vlNbEnregistrements)
   POSITION MESSAGE(5;2)
   MESSAGE ("Traitement de l'enregistrement "+Chaine($vlEnregistrement)+
                                                  Caractere(Retour chariot))
      ` Faire quelque chose avec les enregistrements
   ENREGISTREMENT SUIVANT([touteTable])
   POSITION MESSAGE(5;5)
   $vlReste:=(($vlNbEnregistrements/$vlEnregistrement)-1)*(Heure courante-
                                                  $vhHeureDébut)
   MESSAGE ("Estimation du temps restant : "+Chaine heure($vlReste))
Fin de boucle
FERMER FENETRE
```

Voici le résultat (sous Windows) :



Référence

Creer fenetre, EFFACER FENETRE, FERMER FENETRE, POSITION MESSAGE.

POSITION MESSAGE (x; y)

Paramètre	Type	Description
x	Numérique →	Coordonnée x (horizontale) du curseur
y	Numérique →	Coordonnée y (verticale) du curseur

Description

La commande POSITION MESSAGE est destinée à être utilisée conjointement avec la commande MESSAGE lorsque vous affichez des messages dans une fenêtre ouverte par la commande Creer fenetre.

La commande POSITION MESSAGE détermine l'emplacement du curseur d'insertion des caractères (ce curseur est invivible) : elle définit les coordonnées auxquelles le prochain message s'affichera à l'intérieur de la fenêtre.

L'angle supérieur gauche de la fenêtre représente les coordonnées 0,0. Le curseur est automatiquement positionné à 0,0 lorsqu'une fenêtre est créée ou après l'exécution de la commande EFFACER FENETRE.

Après que POSITION MESSAGE ait défini l'emplacement du curseur, la commande MESSAGE peut être appelée pour afficher des caractères dans la fenêtre.

Conseil : Pour contrôler parfaitement l'affichage des caractères avec les commandes POSITION MESSAGE et MESSAGE, utilisez des polices à espacement constant (par exemple Terminal sous Windows et Monaco sous Mac OS). Dans ces polices, tous les caractères ont la même largeur. Reportez-vous à la description de la commande MESSAGE pour plus d'informations.

Exemples

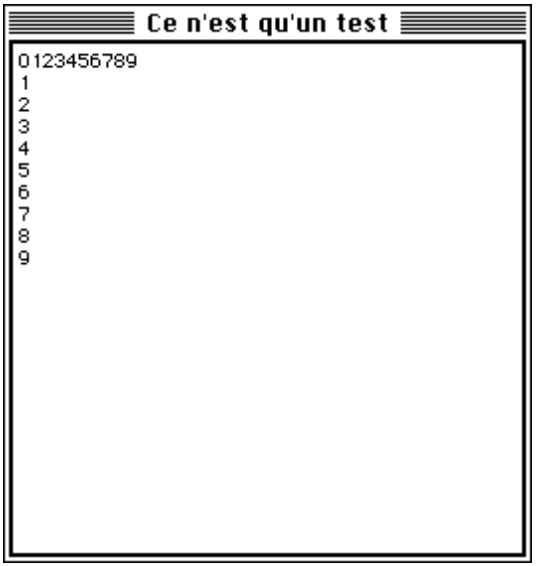
(1) Reportez-vous à l'exemple de la commande MESSAGE.

(2) Reportez-vous à l'exemple de la fonction Nombre de millisecondes.

(3) L'exemple ci-dessous :

```
CREER FENETRE(50;50;300;300;5;"Ce n'est qu'un test")
Boucle ($vlColonne;0;9)
    POSITION MESSAGE($vlColonne;0)
    MESSAGE(Chaine($vlColonne))
Fin de boucle
Boucle ($vlLigne;0;9)
    POSITION MESSAGE(0;$vlLigne)
    MESSAGE(Chaine($vlLigne))
Fin de boucle
$vhHeureDébut:=Heure courante
Repeter
Jusque ((Heure courante-$vhHeureDébut)>†00:00:30†)
```

... affiche la fenêtre suivante (sous Mac OS) pendant 30 secondes :



Référence

MESSAGE.

SUPPRIMER MESSAGES

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Les commandes SUPPRIMER MESSAGES et LAISSER MESSAGES suppriment ou font apparaître les thermomètres de progression affichés par 4D lorsque le programme exécute des opérations de longue durée. Par défaut, les messages sont affichés.

Voici la liste des opérations qui peuvent provoquer l'affichage d'un thermomètre de progression : Application d'une formule, Génération d'un état rapide, Export de données, Import de données, Tri, Génération d'un graphe, Recherche, Recherche par formulaire, Recherche par formule.

Voici les commandes qui peuvent provoquer l'affichage d'un thermomètre de progression :

APPLIQUER A SELECTION
CHERCHER
CHERCHER DANS SELECTION
CHERCHER PAR EXEMPLE
CHERCHER PAR FORMULE
CHERCHER PAR FORMULE DANS SELECTION
ECRITURE DIF
ECRITURE SYLK
EXPORTER TEXTE
GRAPHE SUR SELECTION
IMPORTER TEXTE
JOINTURE
LECTURE DIF
LECTURE SYLK
Max
Min
Moyenne
QR ETAT
REDUIRE SELECTION
SCAN INDEX

SELECTION RETOUR
Somme
TRIER
TRIER PAR FORMULE
VALEURS DISTINCTES

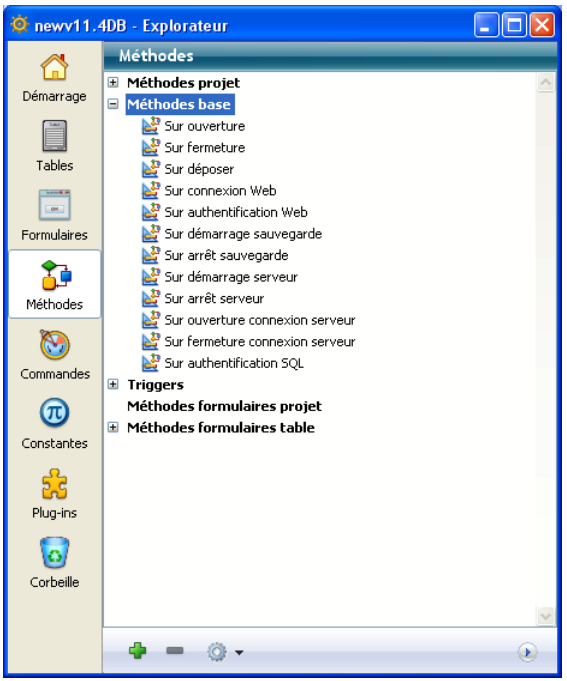
Exemple
L'exemple suivant supprime les thermomètres de progression avant d'effectuer un tri, puis les rétablit après l'opération :

```
SUPPRIMER MESSAGES
TRIER ([Adresses]; [Adresses]CP; >; [Adresses]Nom2; >)
LAISSER MESSAGES
```

40

Méthodes base

Les méthodes base sont des méthodes automatiquement exécutées par 4D lors d'un événement affectant la session dans sa généralité.



Pour créer, ouvrir ou éditer une méthode base :

1. Ouvrez la fenêtre de l'**Explorateur**.
2. Sélectionnez la page **Méthodes**.
3. Déployez le thème **Méthodes base**.
4. Double-cliquez sur la méthode.

ou bien :

4. Sélectionnez la méthode.
5. Appuyez sur la touche **Entrée** ou **Retour chariot**.

Vous éditez une méthode base de la même manière que n'importe quelle autre méthode.

Vous ne pouvez pas appeler une méthode base depuis une autre méthode. Les méthodes base sont automatiquement exécutées par 4D à certains moments de la session de travail. Le tableau suivant résume l'exécution des méthodes base :

Méthode base	4D local	4D Server	4D distant
Sur ouverture	Oui, une fois	Non	Oui, une fois
Sur fermeture	Oui, une fois	Non	Oui, une fois
Sur déposer	Oui, plusieurs fois	Non	Oui, plusieurs fois
Sur authentification Web	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur connexion Web	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur démarrage sauvegarde	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur arrêt sauvegarde	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois
Sur démarrage serveur	Non	Oui, une fois	Non
Sur arrêt serveur	Non	Oui, une fois	Non
Sur ouverture connexion serveur	Non	Oui, plusieurs fois	Non
Sur fermeture connexion serveur	Non	Oui, plusieurs fois	Non
Sur authentification SQL	Oui, plusieurs fois	Oui, plusieurs fois	Oui, plusieurs fois

Pour plus de détails sur chaque méthode base, reportez-vous aux sections suivantes :

- Méthode base Sur ouverture
- Méthode base Sur fermeture
- Méthode base Sur déposer
- Méthode base Sur authentification Web
- Méthode base Sur connexion Web
- Méthode base Sur démarrage sauvegarde
- Méthode base Sur arrêt sauvegarde
- Méthode base Sur authentification SQL
- Méthode base Sur démarrage serveur (cf. Guide de référence de 4D Server)
- Méthode base Sur arrêt serveur (cf. Guide de référence de 4D Server)
- Méthode base Sur ouverture connexion serveur (cf. Guide de référence de 4D Server)
- Méthode base Sur fermeture connexion serveur (cf. Guide de référence de 4D Server)

La Méthode base Sur fermeture est appelée une fois lorsque vous quittez la base.

Les environnements 4D suivants sont concernés :
• 4D en mode local
• 4D en mode distant
• Application 4D compilée et fusionnée avec 4D Volume Desktop

Note : La Méthode base Sur fermeture n'est PAS exécutée par 4D Server.

La Méthode base Sur fermeture est exécutée automatiquement par 4D ; à la différence des méthodes projet, vous ne pouvez pas appeler cette méthode base par programmation. Vous pouvez toutefois l'exécuter depuis l'éditeur de méthodes. Vous pouvez également utiliser des sous-routines.

On sort de la base lorsque l'un des événements suivants se produit :
• L'utilisateur sélectionne la commande **Quitter** dans le menu **Fichier** en mode Développement ou Application (action standard Quitter).
• Un appel à la commande QUITTER 4D a eu lieu.
• Un plug-in 4D a fait appel au point d'entrée QUITTER 4D.

Quel que soit le moyen par lequel la base a été quittée, 4D accomplit les actions qui suivent :

• S'il n'existe pas de méthode base Sur fermeture, 4D détruit chaque process un par un, sans distinction. Si un utilisateur est en train de saisir des données, les enregistrements ne seront pas sauvegardés.

• S'il existe une méthode base Sur fermeture, 4D démarre l'exécution de cette méthode dans un process local nouvellement créé. Vous pouvez ainsi utiliser cette méthode base pour informer d'autres process, via la communication interprocess, qu'ils doivent être fermés (en cas de saisie de données) ou stopper leur exécution. Notez que 4D quittera en tout état de cause — la méthode base Sur fermeture peut assurer le nettoyage et la fermeture de toutes les opérations que vous voulez, mais la fermeture de la base est inéluctable.

La Méthode base Sur fermeture est l'emplacement idéal pour :
• Stopper les process automatiquement démarrés à l'ouverture de la base.
• Sauvegarder (localement, sur disque) les préferences ou paramétrages devant être réutilisés lors de la prochaine session dans la méthode base Sur ouverture.

• Accomplir toute autre action que vous souhaitez exécuter automatiquement à chaque fermeture de la base.

Note: Rappelez-vous que le process créé pour la Méthode base Sur fermeture est un process client (local), qui n'existe donc pas sur le poste serveur. Par conséquent, si vous effectuez dans cette méthode base une recherche ou un tri, tout poste client qui tentera de quitter l'application restera "bloqué". Si vous avez besoin d'accéder aux données lorsque le client quitte, vous devez créer depuis cette méthode base un process global qui, lui, pourra accéder aux données. Dans ce cas toutefois, veillez à ce que ce process puisse terminer son exécution (par l'intermédaire de variables interprocess, par exemple) avant d'être stoppé par la Méthode base Sur fermeture.

Exemple

L'exemple ci-dessous liste les méthodes utilisées dans une base qui note les événements significatifs se produisant lors d'une session de travail. Les étapes sont écrites dans un document texte appelé “Journal”.

• La Méthode base Sur ouverture initialise la variable interprocess <>vbQuit4D, qui signale tous les process utilisés, qu'on sorte ou non de la base. Elle crée aussi le fichier journal, s'il n'existe pas déjà.

```
    ` Méthode base Sur ouverture
C_TEXTE(<>vtIPMessage)
C_BOOLEEN(<>vbQuit4D)
<>vbQuit4D:=Faux

Si (Tester chemin acces("Journal") # Est un document)
    $vhDocRef:=Creer document("Journal")
    Si (OK=1)
        FERMER DOCUMENT($vhDocRef)
    Fin de si
Fin de si
ECRIRE JOURNAL ("Ouverture Session")
```

• La méthode projet ECRIRE JOURNAL, utilisée comme sous-routine par les autres méthodes, écrit l'information qu'elle reçoit dans le fichier journal :

```
    ` Méthode Projet ECRIRE JOURNAL
    ` ECRIRE JOURNAL ( Texte )
    ` ECRIRE JOURNAL ( Description Evenement )
C_TEXTE($1)
C_HEURE($vhDocRef)
```

```
Tant que (Semaphore("$Journal"))
    ENDORMIR PROCESS(Numero du process courant;1)
Fin tant que
$vhDocRef:=Ajouter a document("Journal")
Si (OK=1)
    INFORMATIONS PROCESS(Numero du process courant;$vsProcessNom;$vlEtat;
                                                    $vlTempsEcoule;$vbVisible)
    ENVOYER PAQUET($vhDocRef;Chaine(Date du jour)+Caractere(9)+
            Chaine(Heure courante)+Caractere(9)+Chaine(Numero du process courant)+
                    Caractere(9)+$vsProcessNom+Caractere(9)+$1+Caractere(13))
    FERMER DOCUMENT($vhDocRef)
Fin de si
EFFACER SEMAPHORE("$Journal")
```

Notez que le document est ouvert et refermé à chaque fois. Notez aussi l'emploi d'un sémaphore comme “protection d'accès” au document — nous ne voulons pas que deux process essaient d'accéder au fichier journal en même temps.

• La méthode projet M_AJOUT_ENRG est exécutée lorsque la commande de menu **Ajouter enregistrement** est sélectionnée en mode Application :

```
    ` Méthode Projet M_AJOUT_ENRG

CHANGER BARRE(1)
Repeter
    AJOUTER ENREGISTREMENT([Table1];*)
    Si (OK=1)
        ECRIRE JOURNAL ("Ajout d'enregistrement #"+
                            Chaine(Numero enregistrement([Table1]))+" dans Table1")
    Fin de si
Jusque ((OK=0) | <>vbQuit4D)
```

Cette méthode effectue une boucle jusqu'à ce que l'utilisateur annule la saisie de données ou que la base soit refermée.

• Le formulaire entrée de la [Table1] inclut le traitement des événements Sur appel extérieur. Ainsi, même si un process est en saisie de données, on en sort "en douceur", et l'utilisateur peut sauvegarder (ou non) la saisie en cours :

```
    ` Méthode formulaire [Table1];"Entrée"
Au cas ou
        : (Evenement formulaire=Sur appel extérieur)
```

```
            Si (<>vtIPMessage="QUITTER")
                CONFIRMER("Voulez-vous sauvegarder les modifications dans cet
                                                                    enregistrement ?")
                Si (OK=1)
                    VALIDER
                Sinon
                    NE PAS VALIDER
                Fin de si
            Fin de si
        Fin de cas
```

• La méthode projet M_QUIT est exécutée lorsque la commande **Quitter** du menu **Fichier** en mode Application est sélectionnée :

```
    ` Méthode Projet M_QUIT
$vlProcessID:=Nouveau process("ON_QUIT";32*1024;"$ON_QUIT")
```

Cette méthode utilise une astuce. Lorsque la commande QUITTER 4D est appelée, elle a un effet immédiat. En conséquence, le process dans lequel elle est appelée est placé en “mode arrêt”, jusqu'à ce que la base ait été effectivement refermée. Comme ce process peut être un des process dans lequel est effectuée la saisie de données, l'appel à QUITTER 4D est réalisé dans un process local qui n'est démarré que pour ce but. Voici la méthode ON_QUIT:

```
    ` Méthode projet ON_QUIT
CONFIRMER("Etes-vous certain de vouloir quitter ?")
Si (OK=1)
    ECRIRE JOURNAL ("Sortie de la base")
    QUITTER 4D
        ` QUITTER 4D a un effet immédiat. Aucune ligne de code n'est exécutée par la suite.
        ` ...
Fin de si
```

• Enfin, voici la méthode base Sur fermeture, qui signale à tous les process utilisateur qu'“il est temps de partir !”. Elle met <>vbQuit4D à Vrai et envoie des messages interprocess aux process utilisateur qui gèrent la saisie de données :

```
    ` Méthode base Sur fermeture
<>vbQuit4D:=Vrai
Repeter
    $vbfini:=Vrai
```

```
        Boucle ($vlProcess;1;Nombre de process)
            INFORMATIONS PROCESS($vlProcess;$vsProcessNom;$vlEtat;$vlTempsEcoule;
                                                                    $vbVisible)
            Si ((($vsProcessNom="ML_@") | ($vsProcessNom="M_@"))) & ($vlEtat>=0))
                $vbfini:=Faux
                <>vtIPMessage:="QUITTER"
                PASSER AU PREMIER PLAN($vlProcess)
                APPELER PROCESS($vlProcess)
                $vhStart:=Heure courante
                Repeter
                    ENDORMIR PROCESS(Numero du process courant;60)
                Jusque ((Statut du process($vlProcess)<0) | ((Heure courante-$vhStart)>=
                                                                    ?00:01:00?))
            Fin de si
        Fin de boucle
    Jusque ($vbfini)
    ECRIRE JOURNAL ("Fermeture de session")
```

Note : Les process dont les noms commencent par "ML_..." ou "M_..." sont démarrés par les commandes de menus pour lesquelles la propriété **Démarrer un process** a été sélectionnée. Dans cet exemple, ce sont les process démarrés suite à la sélection de la commande de menu **Ajouter enregistrement**.

Le test (Heure courante-$vhStart)>=?00:01:00? permet à la méthode base de sortir de la boucle “en attente de l'autre process”, si l'autre process ne réagit pas pendant une minute.

• Voici un exemple type de fichier Journal produit par la base :

2/6/03	15:47:25	1	Process principal	Ouverture de Session
2/6/03	15:55:43	5	ML_1	Ajout d'enregistrement #23 dans Table1
2/6/03	15:55:46	5	ML_1	Ajout d'enregistrement #24 dans Table1
2/6/03	15:55:54	6	$On_QUIT	Sortie de la base
2/6/03	15:55:58	7	$xx	Fermeture de session

Note : $xx est le nom du process local démarré par 4D pour exécuter la méthode base Sur fermeture.

Référence

Méthode base Sur ouverture, QUITTER 4D.

La Méthode base Sur ouverture est exécutée une seule fois, au moment de l'ouverture de la base.

Les environnements 4D suivants sont concernés :
• 4D en mode local
• 4D en mode distant (sur le poste client une fois que la connexion a été acceptée par 4D Server)
• Application 4D compilée et fusionnée avec 4D VolumeDesktop

Note : La Méthode base Sur ouverture n'est PAS exécutée par 4D Server.

La méthode base Sur ouverture est exécutée automatiquement par 4D ; à la différence des méthodes projet, vous ne pouvez pas appeler cette méthode base par programmation. Vous pouvez toutefois l'exécuter depuis l'éditeur de méthodes. Vous pouvez également utiliser des sous-routines.

La méthode base Sur ouverture est l'emplacement idéal pour :
• Initialiser les variables interprocess que vous utiliserez pendant toute la session de travail.
• Démarrer automatiquement des process à l'ouverture de la base.
• Charger des préférences ou des paramétrages sauvegardés dans ce but lors de la session de travail précédente.
• Empêcher l'ouverture de la base si une condition n'est pas remplie (comme par exemple, une ressource système manquante) par l'appel explicite de la commande QUITTER 4D.
• Accomplir toute autre action que vous souhaitez exécuter automatiquement à chaque ouverture de la base.

Exemple
Reportez-vous à l'exemple de la section Méthode base Sur fermeture.

Référence

Méthode base Sur fermeture, Méthodes, Présentation des méthodes base, QUITTER 4D.

41

Opérateurs

Les opérateurs sont des symboles permettant d'effectuer des opérations sur des expressions. Les opérateurs peuvent effectuer des calculs sur des nombres, des dates et des heures. Ils effectuent aussi des opérations logiques sur les chaînes, les booléens et des expressions logiques ainsi que des opérations spéciales sur des images. Ils combinent des expressions simples pour générer de nouvelles expressions.

Priorité
L'ordre dans lequel une expression est évaluée s'appelle la priorité. 4D applique strictement une règle de priorité de gauche à droite. **L'ordre algébrique n'est pas appliqué**. Par exemple :

```
3 + 4 * 5
```

retourne 35 car l'expression est évaluée comme 3 + 4, qui donne 7, multiplié par 5, ce qui donne 35.

Les parenthèses doivent être utilisées pour forcer l'ordre de calcul en fonction de vos besoins. Par exemple :

```
3 + (4 * 5)
```

retourne 23 car l'expression (4 * 5) est évaluée en premier lieu. Le résultat (20) est alors ajouté à 3, ce qui donne le résultat final 23.

Des parenthèses peuvent être incluses dans d'autres parenthèses. Assurez-vous qu'il y ait une parenthèse fermante pour chaque parenthèse ouverte. Une parenthèse manquante ou placée à un mauvais endroit peut soit donner un résultat erroné, soit renvoyer une expression invalide. De plus, si vous avez l'intention de compiler vos applications, vous devez vous assurer d'une bonne utilisation des parenthèses. Le compilateur interprètera toute parenthèse manquante ou superflue comme une erreur de syntaxe.

L'opérateur d'affectation (ou d'assignation)
L'opérateur d'affectation := se distingue des autres opérateurs. Au lieu de combiner des expressions en une seule, l'opérateur d'affectation copie la valeur de l'expression située à sa droite dans la variable ou le champ qui se trouve à sa gauche. Par exemple, la ligne suivante place la valeur 4 (le nombre de caractères présents dans le mot Pont) dans la variable maVar, qui prend alors le type numérique.

```
maVar := Longueur ("Pont")
```

Important : Ne confondez pas l'opérateur d'affectation := avec l'opérateur de comparaison d'égalité =.

Les autres opérateurs proposés par le langage de 4D sont décrits dans les sections suivantes :

Opérateurs sur les chaînes
Référez-vous à la section Opérateurs sur les chaînes.

Opérateurs numériques
Référez-vous à la section Opérateurs numériques.

Opérateurs sur les dates
Référez-vous à la section Opérateurs sur les dates.

Opérateurs sur les heures
Référez-vous à la section Opérateurs sur les heures.

Comparateurs
Référez-vous à la section Opérateurs de comparaison.

Opérateurs logiques
Référez-vous à la section Opérateurs logiques.

Opérateurs sur les images
Référez-vous à la section Opérateurs sur les images.

Opérateurs sur les bits
Référez-vous à la section Opérateurs sur les bits.

Opérateurs de comparaison

version 11.1 (Modifiée)

Les tableaux suivants décrivent les opérateurs de comparaison. Ces opérateurs peuvent être appliqués aux expressions de type chaîne, numérique, date, heure et pointeur (il n'est donc pas possible de les utiliser avec des expressions de type tableau, image ou BLOB).
Une expression qui utilise un opérateur de comparaison retourne une valeur booléenne, soit VRAI soit FAUX.

Comparaisons de chaînes

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Chaîne = Chaîne	Booléen	"abc" = "abc" "abc" = "abd"	Vrai Faux
Inégalité	Chaîne # Chaîne	Booléen	"abc" # "abd" "abc" # "abc"	Vrai Faux
Supérieur à	Chaîne > Chaîne	Booléen	"abd" > "abc" "abc" > "abc"	Vrai Faux
Inférieur à	Chaîne < Chaîne	Booléen	"abc" < "abd" "abc" < "abc"	Vrai Faux
Supérieur ou égal à	Chaîne >= Chaîne	Booléen	"abd" >= "abc" "abc" >= "abd"	Vrai Faux
Inférieur ou égal à	Chaîne <= Chaîne	Booléen	"abc" <= "abd" "abd" <= "abc"	Vrai Faux
Contient mot-clé	Chaîne % Chaîne	Booléen	"Alpha Bravo" % "Bravo" "Alpha Bravo" % "ravo"	Vrai Faux

Important : Des informations supplémentaires sur les comparaisons de chaînes sont fournies à la fin de cette section.

Comparaisons de numériques

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Nombre = Nombre	Booléen	10 = 10 10 = 11	Vrai Faux
Inégalité	Nombre # Nombre	Booléen	10 #11 10 # 10	Vrai Faux
Supérieur à	Nombre > Nombre	Booléen	11 > 10 10 > 11	Vrai Faux
Inférieur à	Nombre < Nombre	Booléen	10 < 11 11 < 10	Vrai Faux
Supérieur ou égal à	Nombre >= Nombre	Booléen	11 >= 10 10 >= 11	Vrai Faux
Inférieur ou égal à	Nombre <= Nombre	Booléen	10 <= 11 11 <= 10	Vrai Faux

Comparaisons de dates

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Date = Date	Booléen	!1/1/97! =!1/1/97! !20/1/97! =!1/1/97!	Vrai Faux
Inégalité	Date # Date	Booléen	!20/1/97! # !1/1/97! !1/1/97! # !1/1/97!	Vrai Faux
Supérieur à	Date > Date	Booléen	!20/1/97! > !1/1/97! !1/1/97! > !1/1/97!	Vrai Faux
Inférieur à	Date < Date	Booléen	!1/1/97! < !20/1/97! !1/1/97! < !1/1/97!	Vrai Faux
Supérieur ou égal à	Date >= Date	Booléen	!20/1/97! >=!1/1/97! !1/1/97!>=!20/1/97!	Vrai Faux
Inférieur ou égal à	Date <= Date	Booléen	!1/1/97!<=!20/1/97! !20/1/97!<=!1/1/97!	Vrai Faux

Comparaisons d'heures

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Heure = Heure	Booléen	?01:02:03? = ?01:02:03? ?01:02:03? = ?01:02:04?	Vrai Faux
Inégalité	Heure # Heure	Booléen	?01:02:03? # ?01:02:04? ?01:02:03? # ?01:02:03?	Vrai Faux
Supérieur à	Heure > Heure	Booléen	?01:02:04? > ?01:02:03? ?01:02:03? > ?01:02:03?	Vrai Faux
Inférieur à	Heure < Heure	Booléen	?01:02:03? < ?01:02:04? ?01:02:03? < ?01:02:03?	Vrai Faux
Supérieur ou égal à	Heure >= Heure	Booléen	?01:02:03? >=?01:02:03? ?01:02:03? >=?01:02:04?	Vrai Faux
Inférieur ou égal à	Heure <= Heure	Booléen	?01:02:03? <=?01:02:03? ?01:02:04? <=?01:02:03?	Vrai Faux

Comparaisons de pointeurs

Avec :

```
    ` vPtrA et vPtrB pointent sur le même objet
vPtrA:=->unObjet
vPtrB:=->unObjet
    ` vPtrC pointe sur un autre objet
vPtrC:=->autreObjet
```

Opération	Syntaxe	Retourne	Expression	Valeur
Egalité	Pointeur = Pointeur	Booléen	vPtrA = vPtrB vPtrA = vPtrC	Vrai Faux
Inégalité	Pointeur # Pointeur	Booléen	vPtrA # vPtrC vPtrA # vPtrB	Vrai Faux

Remarques sur les comparaisons de chaînes

Voici quelques informations supplémentaires sur les comparaisons d'alphanumériques :

• Les chaînes sont toujours comparées caractère par caractère (hormis en cas de recherche par mot-clé, cf. ci-dessous).

• Lors d'une comparaison de chaînes, 4D ne tient pas compte de la casse des caractères ; par exemple, "a"="A" retourne VRAI. Pour savoir si des caractères sont en majuscules ou en minuscules, vous devez comparer leurs codes de caractères. Par exemple, l'expression suivante retourne FAUX :

Code de caractere ("A") = **Code de caractere** ("a") ` 65 n'est pas égal à 97

• Lors d'une comparaison de chaînes, les caractères diacritiques sont comparés à l'aide de la table de comparaison des caractères de votre machine. Par exemple, les expressions suivantes retournent VRAI :

"n" = "ñ"
"n" = "Ñ"
"A"="å"
` etc

• A la différence des autres comparaisons de chaîne, les recherches par mots-clés recherchent des “mots” dans des “textes” : les mots sont évalués individuellement et dans leur globalité. L’opérateur % retournera toujours Faux si la recherche porte sur plusieurs mots ou une partie de mot (par exemple une syllabe). Les “mots” sont des chaînes de caractères encadrées par des “séparateurs”, qui sont les espaces, les caractères de ponctuation et les tirets. Une apostrophe, comme dans “aujourd'hui” est considérée comme partie du mot. Les nombres peuvent être recherchés car ils sont évalués dans leur ensemble (incluant les symboles décimaux). Les autres symboles (monnaie, température, etc.) seront ignorés.

"Alpha Bravo Charlie" % "Bravo" ` Retourne Vrai
"Alpha Bravo Charlie" % "vo" ` Retourne Faux
"Alpha Bravo Charlie" % "Alpha Bravo" ` Retourne Faux
"Alpha,Bravo,Charlie" % "Alpha" ` Retourne Vrai
"Software and Computers" % "comput@" ` Retourne Vrai

Note : Pour plus d'informations sur les règles de prise en compte des mots-clés, reportez-vous à l'adresse http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries.

• Le joker (@) peut être utilisé dans toute comparaison de chaînes. Il remplace un ou plusieurs caractères. Ainsi, par exemple, l'expression suivante est évaluée à VRAI :

"abcdefghij" = "abc@"

Le joker doit être utilisé dans le second opérande (la chaîne qui se trouve à droite de l'opérateur). L'expression suivante est évaluée à FAUX car le joker est alors considéré en tant que caractère :

"abc@" = "abcdefghij"

Le joker signifie “un ou plusieurs caractères sinon rien”. Les expressions suivantes sont évaluées à VRAI :

```
"abcdefghij" = "abcdefghij@"
"abcdefghij" = "@abcdefghij"
"abcdefghij" = "abcd@efghij"
"abcdefghij" = "@abcdefghij@"
"abcdefghij" = "@abcde@fghij@"
```

En revanche, dans tous les cas, lorsque deux jokers consécutifs sont placés dans une comparaison de chaînes, celle-ci sera évaluée à FAUX. L'expression suivante est à FAUX :

```
"abcdefghij" = "abc@@fg"
```

Lorsque l'opérateur de comparaison est ou contient un symbole < ou >, seule la comparaison avec un seul joker situé en fin d'opérande est prise en charge :

```
"abcd" <= "abc@" `Comparaison valide
"abcd" <= "abc@ef" `Comparaison non valide
```

Note
Si vous souhaitez effectuer des comparaisons ou des recherches utilisant @ en tant que caractère (et non en tant que joker), vous disposez de deux possibilités :

• Utiliser l'instruction Code de caractere(Arobase).
Imaginons par exemple que vous souhaitiez savoir si une chaîne se termine par le caractère @.
- l’expression suivante (si $vaValeur n'est pas vide) retourne toujours VRAI :

```
($vaValeur[[Longueur($vaValeur)]]="@")
```

- l'expression suivante sera correctement évaluée :

```
(Code de caractere($vaValeur[[Longueur($vaValeur)]])#64)
```

• Utiliser l'option “Considérer @ comme joker uniquement au début et à la fin des chaînes de caractères”, accessible via la boîte de dialogue des Préférences.
Cette option permet de paramétrer le mode d’interprétation du caractère @ lorsque celui-ci est inclus dans une chaîne de caractères. Elle peut donc influer sur le fonctionnement des opérateurs de comparaison utilisés dans le cadre de recherches ou de tris. Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Référence

Opérateurs, Opérateurs logiques, Opérateurs numériques, Opérateurs sur les bits, Opérateurs sur les dates, Opérateurs sur les heures, Opérateurs sur les images.

4D supporte deux opérateurs logiques : l'opérateur d'intersection (ET) et l'opérateur de réunion inclusive (OU). Ces deux opérateurs ne s'appliquent qu'aux expressions booléennes. Le ET logique retourne VRAI si les deux expressions sont VRAIES. Le OU logique retourne VRAI si au moins une des expressions est VRAIE.

4D vous permet également d'exploiter les fonctions booléennes Vrai, Faux et Non. Pour plus d'informations, reportez-vous aux descriptions de ces commandes.

Le tableau suivant décrit les opérateurs logiques :

Opération	Syntaxe	Retourne	Expression	Valeur
ET	Booléen & Booléen	Booléen	("A" = "A") & (15 # 3)	Vrai
			("A" = "B") & (15 # 3)	Faux
			("A" = "B") & (15 = 3)	Faux
OU	Booléen | Booléen	Booléen	("A" = "A") | (15 # 3)	Vrai
			("A" = "B") | (15 # 3)	Vrai
			("A" = "B") | (15 = 3)	Faux

Voici la "table de vérité" pour l'opérateur logique "ET" :

Expr1	Expr2	Expr1 & Expr2
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Voici la "table de vérité" pour l'opérateur logique "OU" :

Expr1	Expr2	Expr1 | Expr2
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Astuce
Si vous devez calculer une réunion exclusive (le "OU" exclusif) entre Expr1 et Expr2, écrivez :

(Expr1 | Expr2) & **Non**(Expr1 & Expr2)

Référence

Opérateurs, Opérateurs de comparaison, Opérateurs numériques, Opérateurs sur les bits, Opérateurs sur les chaînes, Opérateurs sur les dates, Opérateurs sur les heures, Opérateurs sur les images.

Opérateurs numériques

Une expression qui utilise un opérateur numérique retourne une valeur numérique. Le tableau suivant décrit les opérateurs numériques :

Opération	Syntaxe	Retourne	Expression	Valeur
Addition	Nombre + Nombre	Nombre	2 + 3	5
Soustraction	Nombre – Nombre	Nombre	3 – 2	1
Multiplication	Nombre * Nombre	Nombre	5 * 2	10
Division	Nombre /Nombre	Nombre	5 / 2	2.5
Division entière	Nombre \ Nombre	Nombre	5 \ 2	2
Modulo	Nombre % Nombre	Nombre	5 % 2	1
Exponentiation	Nombre ^ Nombre	Nombre	2 ^ 3	8

L'opérateur modulo % divise le premier nombre par le second et retourne le reste de la division entière. Voici quelques exemples :

- 10 % 2 retourne 0 car la division de 10 par 2 ne donne pas de reste.
- 10 % 3 retourne 1 car le reste est 1.
- 10,5 % 2 retourne 0 car le reste n'est pas un nombre entier.

ATTENTION :
- L'opérateur modulo % retourne des valeurs significatives avec des nombres appartenant à la catégorie des entiers longs (de –2^31 à +2^31 moins 1). Pour calculer le modulo de nombres qui ne sont pas dans cet intervalle, utilisez la fonction Modulo.
- L'opérateur division entière \ retourne des valeurs significatives avec des nombres entiers uniquement.

Référence

Opérateurs, Opérateurs de comparaison, Opérateurs logiques, Opérateurs sur les bits, Opérateurs sur les chaînes, Opérateurs sur les dates, Opérateurs sur les heures, Opérateurs sur les images.

Opérateurs sur les bits

Les opérateurs sur les bits s'appliquent à des expressions ou valeurs de type Entier long.

Note : Si vous passez une valeur de type Entier ou Numérique (Réel) à un opérateur sur les bits, 4D la convertit en Entier long avant de calculer le résultat de l'expression.

Lorsque vous employez des opérateurs sur les bits, vous devez considérer une valeur de type Entier long comme un tableau de 32 bits. Les bits sont numérotés de 0 à 31, de droite à gauche.

Comme un bit peut valoir 0 (zéro) ou 1, vous pouvez également considérer une valeur de type Entier long comme une expression dans laquelle vous pouvez stocker 32 valeurs de type Booléen. Lorsque le bit vaut 1, la valeur est Vrai et lorsque le bit vaut 0, la valeur est Faux.

Une expression utilisant un opérateur sur les bits retourne une valeur de type Entier long, à l'exception de l'opérateur Tester bit avec lequel l'expression retournée est du type Booléen. Le tableau suivant fournit la liste des opérateurs sur les bits et leur syntaxe :

Opération	Opérateur	Syntaxe	Retourne	
ET	&	E.long & E.long	E.long	
OU (inclusif)	|	E.long | E.long	E.long	
OU (exclusif)	^|	E.long ^| E.long	E.long	
Décaler bits à gauche	<<	E.long << E.long	E.long	(voir note 1)
Décaler bits à droite	>>	E.long >> E.long	E.long	(voir note 1)
Mettre bit à 1	?+	E.long ?+ E.long	E.long	(voir note 2)
Mettre bit à 0	?-	E.long ?- E.long	E.long	(voir note 2)
Tester bit	??	E.long ?? E.long	Booléen	(voir note 2)

Notes
(1) Dans les opérations utilisant Décaler bits à gauche et Décaler bits à droite, le second opérande indique le nombre de décalages de bits du premier opérande à effectuer dans la valeur retournée. Par conséquent, ce second opérande doit être compris entre 0 et 31. Notez qu'un décalage de 0 retourne une valeur inchangée et qu'un décalage de plus de 31 bits retourne 0x00000000 car tous les bits sont perdus. Si vous passez une autre valeur en tant que second opérande, le résultat sera non significatif.
(2) Dans les opérations utilisant Mettre bit à 1, Mettre bit à 0 et Tester bit, le second opérande indique le numéro du bit sur lequel agir. Par conséquent, ce second opérande doit être compris entre 0 et 31. Sinon, l'expression retourne inchangée la valeur du premier opérande pour Mettre bit à 1 et Mettre bit à 0, et retourne Faux pour Tester bit.

Le tableau suivant dresse la liste des opérateurs sur les bits et de leurs effets :

Opération sur les bits	Description
ET	Chaque bit retourné est le résultat de l'opération ET logique appliquée aux deux bits opérandes. Voici la table du ET logique : 1 & 1 → 1 0 & 1 → 0 1 & 0 → 0 0 & 0 → 0 En résumé, le résultat vaut 1 si les deux bits opérandes valent 1, dans tous les autres cas le bit résultant vaut 0.
OU (inclusif)	Chaque bit retourné est le résultat de l'opération OU inclusif logique appliquée aux deux bits opérandes. Voici la table du OU inclusif logique : 1 | 1 → 1 0 | 1 → 1 1 | 0 → 1 0 | 0 → 0 En résumé, le résultat vaut 1 si au moins un des deux bits opérandes vaut 1, sinon le bit résultant vaut 0.
OU (exclusif)	Chaque bit retourné est le résultat de l'opération OU exclusif logique appliquée aux deux bits opérandes. Voici la table du OU exclusif logique: 1 ^| 1 → 0 0 ^| 1 → 1 1 ^| 0 → 1 0 ^| 0 → 0 Donc, le résultat vaut 1 si un seul des deux bits opérandes vaut 1 (et pas l'autre), dans tous les autres cas le bit résultant vaut 0.
Décaler bits à gauche	La valeur retournée correspond au premier opérande dont la valeur est décalée vers la gauche du nombre de bits spécifié par le second opérande. Les bits auparavant situés à gauche sont perdus et les nouveaux bits situés à droite ont la valeur 0. **Note** : En ne tenant compte que des valeurs positives, un décalage vers la gauche de N bits revient à multiplier la valeur par 2^N.

Décaler bits à droite	La valeur retournée correspond au premier opérande dont la valeur est décalée vers la droite du nombre de bits spécifié par le second opérande. Les bits auparavant situés à droite sont perdus et les nouveaux bits situés à gauche ont la valeur 0. **Note** : En ne tenant compte que des valeurs positives, un décalage vers la droite de N bits revient à diviser la valeur par 2^N.
Mettre bit à 1	La valeur retournée est la valeur du premier opérande dans lequel le bit dont le numéro est spécifié par le second opérande est positionné à 1. Les autres bits sont inchangés.
Mettre bit à 0	La valeur retournée est la valeur du premier opérande dans lequel le bit dont le numéro est spécifié par le second opérande est positionné à 0. Les autres bits sont inchangés.
Tester bit	Retourne Vrai si, dans le premier opérande, le bit dont le numéro est indiqué par le second opérande vaut 1. Retourne Faux si, dans le premier opérande, le bit dont le numéro est indiqué par le second opérande vaut 0.

Exemples

(1) Le tableau ci-dessous propose un exemple d'utilisation de chaque opérateur sur les bits :

Opération	Exemple	Résultat
ET	0x0000FFFF & 0xFF00FF00	0x0000FF00
OU (inclusif)	0x0000FFFF | 0xFF00FF00	0xFF00FFFF
OU (exclusif)	0x0000FFFF ^| 0xFF00FF00	0xFF0000FF
Décaler bit gauche	0x0000FFFF << 8	0x00FFFF00
Décaler bit droit	0x0000FFFF >> 8	0x000000FF
Mettre bit à 1	0x00000000 ?+ 16	0x00010000
Mettre bit à 0	0x00010000 ?- 16	0x00000000
Tester bit	0x00010000 ?? 16	Vrai

(2) 4D exploite de nombreuses constantes prédéfinies. Le nom de certaines d'entre elles commence par "Bit" ou "Masque". C'est le cas des constantes incluses dans le thème Propriétés des ressources :

Constante	Type	Valeur
Masque ressource heap système	Entier long	64
Bit ressource heap système	Entier long	6
Masque ressource purgeable	Entier long	32

Bit ressource purgeable	Entier long	5
Masque ressource verrouillée	Entier long	16
Bit ressource verrouillée	Entier long	4
Masque ressource protégée	Entier long	8
Bit ressource protégée	Entier long	3
Masque ressource préchargée	Entier long	4
Bit ressource préchargée	Entier long	2
Masque ressource modifiée	Entier long	2
Bit ressource modifiée	Entier long	1

Ces constantes vous permettent de tester la valeur retournée par la fonction Lire proprietes ressource ou de définir la valeur à passer à ECRIRE PROPRIETES RESSOURCE. Les constantes dont le nom débute par “Bit” fournissent la position du bit que vous voulez tester, effacer ou fixer. Les constantes dont le nom débute par “Masque” sont des valeurs de type Entier long dans lesquelles seul le bit que vous voulez tester, effacer ou fixer est égal à 1.

Si, par exemple, vous devez tester si une ressource (dont les propriétés sont stockées dans la variable $vlResAttr) est purgeable ou non, vous pouvez écrire :

```
Si ($vlResAttr ?? Bit ressource purgeable) ` Est-ce que la ressource est purgeable?
```

ou :

```
Si (($vlResAttr & Masque ressource purgeable) # 0) ` Est-ce que la ressource est purgeable?
```

A l'inverse, vous pouvez utiliser ces constantes pour définir le même bit. Vous pouvez écrire :

```
$vlResAttr:=$vlResAttr ?+ Bit ressource purgeable
```

ou :

```
$vlResAttr:=$vlResAttr | Bit ressource purgeable
```

(3) Vous voulez stocker deux valeurs entières dans un Entier long. Vous pouvez écrire :

```
$vlLong:=($viIntA<<16) | $viIntB ` Stocker deux Entiers dans un Entier long
$vlIntA:=$vlLong>>16 ` Extraire l'Entier stocké dans le mot haut
$viIntB:=$vlLong & 0xFFFF ` Extraire l'entier stocké dans le mot bas
```

Note : Soyez prudent lorsque vous manipulez des Entiers longs ou des Entiers avec des expressions qui combinent des opérateurs sur les bits et des opérateurs numériques. Le bit supérieur (bit 31 pour un Entier long, bit 15 pour un Entier) détermine le signe de la valeur (positif s'il est à 0, négatif s'il est à 1). Les opérateurs numériques utilisent ce bit pour détecter le signe d'une valeur, mais les opérateurs sur les bits n'en tiennent pas compte.

Référence

Opérateurs, Opérateurs de comparaison, Opérateurs logiques, Opérateurs numériques, Opérateurs sur les chaînes, Opérateurs sur les dates, Opérateurs sur les heures, Opérateurs sur les images.

Opérateurs sur les chaînes

Une expression qui utilise un opérateur sur les chaînes alphanumériques retourne une chaîne. Le tableau suivant décrit les opérateurs sur les chaînes :

Opération	Syntaxe	Retourne	Expression	Valeur
Concaténation	Chaîne + Chaîne	Chaîne	"abc" + "def"	"abcdef"
Répétition	Chaîne * Nombre	Chaîne	"ab" * 3	"ababab"

Référence

Opérateurs, Opérateurs de comparaison, Opérateurs logiques, Opérateurs numériques, Opérateurs sur les bits, Opérateurs sur les dates, Opérateurs sur les heures, Opérateurs sur les images.

Opérateurs sur les dates

Une expression qui utilise un opérateur sur les dates retourne une date ou une valeur numérique, suivant l'opération effectuée. Toutes les opérations sur les dates retournent des valeurs exactes, tenant compte en particulier des années bissextiles. Le tableau suivant décrit les opérateurs sur les dates :

Opération	Syntaxe	Retourne	Expression	Valeur
Différence	Date – Date	Nombre	!20/1/97! – !1/1/97!	19
Addition	Date + Nombre	Date	!20/1/97! + 9	!29/1/97!
Soustraction	Date – Nombre	Date	!20/1/97! – 9	!11/1/97!

Référence

Opérateurs, Opérateurs de comparaison, Opérateurs logiques, Opérateurs numériques, Opérateurs sur les bits, Opérateurs sur les chaînes, Opérateurs sur les heures, Opérateurs sur les images.

Une expression qui utilise un opérateur sur les heures retourne une heure ou une valeur numérique, suivant l'opération effectuée. Le tableau suivant décrit les opérateurs sur les heures :

Opération	Syntaxe	Retourne	Expression	Valeur
Addition	Heure + Heure	Heure	?02:03:04? + ?01:02:03?	?03:05:07?
Soustraction	Heure – Heure	Heure	?02:03:04? – ?01:02:03?	?01:01:01?
Addition	Heure + Nombre	Nombre	?02:03:04? + 65	7449
Soustraction	Heure – Nombre	Nombre	?02:03:04? – 65	7319
Multiplication	Heure * Nombre	Nombre	?02:03:04? * 2	14768
Division	Heure / Nombre	Nombre	?02:03:04? / 2	3692
Division entière	Heure \ Nombre	Nombre	?02:03:04? \ 2	3692
Modulo	Heure % Nombre	Nombre	?02:03:04? % 2	0

Astuces

(1) Pour obtenir une expression de type heure à partir d'une expression qui combine une heure avec un chiffre, utilisez les fonctions Heure et Chaine heure. Par exemple :

```
` La ligne suivante assigne à la variable $vlSecondes le nombre de secondes qui, dans
` une heure à partir demaintenant, se seront écoulées depuis minuit
$vlSecondes:=Heure courante+3600

` La ligne suivante assigne à la variable $vhBientôt l'heure qu'il sera dans une heure
$vhBientôt:=Heure(Chaine heure(Heure courante+3600))
```

La seconde ligne peut également être écrite de la façon suivante :

```
` La ligne suivante affecte dans la variable $vhBientôt l'heure qu'il sera dans une heure
$vhBientôt:=Heure courante+?01:00:00?
```

Vous pouvez utiliser cette astuce si, lors du développement de votre application, vous vous retrouvez dans la situation où un délai exprimé en secondes doit être ajouté à une valeur de type heure disponible en tant que valeur numérique.

(2) Il faut parfois convertir une expression heure en expression numérique. Par exemple, vous ouvrez un document sur disque à l'aide de la fonction Ouvrir document, qui retourne un numéro de référence de document (DocRef) qui est une expression de type heure. Vous pouvez passer DocRef à une routine de plug-in 4D qui attend une valeur numérique comme numéro de référence de document. Dans ce cas, ajoutez 0 (zéro) à l'heure pour obtenir une valeur numérique, sans la modifier. Par exemple :

```
    ` Sélectionner et ouvrir un document
$vhDocRef:=Ouvrir document("")
Si (OK=1)
        ` Passez l'expression heure DocRef en tant qu'expression numérique à
        ` une routine d'extension 4D
    faire quelque chose (0+$vhDocRef)
Fin de si
```

Référence

Opérateurs, Opérateurs de comparaison, Opérateurs logiques, Opérateurs numériques, Opérateurs sur les bits, Opérateurs sur les chaînes, Opérateurs sur les dates, Opérateurs sur les images.

Opérateurs sur les images

Le tableau suivant décrit les opérateurs que vous pouvez utiliser avec 4D sur les images. Une expression qui utilise un opérateur sur les images retourne toujours une image.

Opération	Syntaxe	Action
Concaténation horizontale	Image1 + Image2	Place Image2 à la droite d'Image1
Concaténation verticale	Image1 / Image2	Place Image2 au-dessous d'Image1
Superposition exclusive	Image1 & Image2	Effectue un OU exclusif entre Image1 et Image2
Superposition inclusive	Image1 | Image2	Effectue un OU inclusif entre Image1 et Image2
Déplacement horizontal	Image + Nombre	Déplace Image horizontalement d'un nombre de pixels égal à Nombre
Déplacement vertical	Image / Nombre	Déplace Image verticalement d'un nombre de pixels égal à Nombre
Redimensionnement	Image * Nombre	Redimensionne Image au pourcentage Nombre
Extension horizontale	Image *+ Nombre	Redimensionne Image horizontalement au pourcentage Nombre
Extension verticale	Image */ Nombre	Redimensionne Image verticalement au pourcentage Nombre

Les deux opérateurs & et | retournent toujours une image de type bitmap, quel que soit le type des deux images source. La raison en est que 4D dessine d'abord les images en mémoire en tant que bitmaps et calcule l'image résultante appliquant l'opérateur OU sur chaque pixel du bitmap.

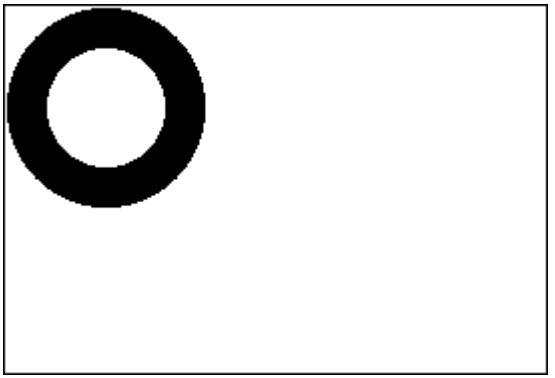
Note : La commande COMBINER IMAGES permet d'effectuer des superpositions en conservant les caractéristiques de chaque image source dans l'image résultante.

Les autres opérateurs sur les images retournent des images vectorielles si les deux images source sont elles aussi vectorielles (rappelez-vous qu'une image imprimée avec le format d'affichage Sur fond est imprimée en tant que bitmap).

Exemples

Toutes les images qui sont affichées utilisent le format d'affichage Image sur fond.

Voici l'image cercle :

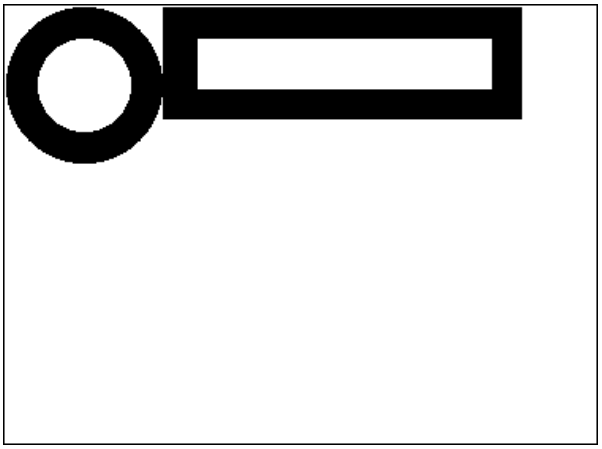


Voici l'image rectangle :

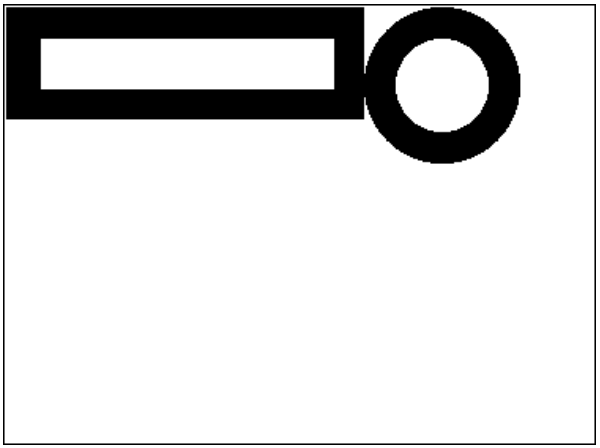
Dans les exemples ci-dessous, chaque expression est suivie de sa représentation graphique.

• Concaténation horizontale

cercle + rectangle ` Placer le rectangle à droite du cercle

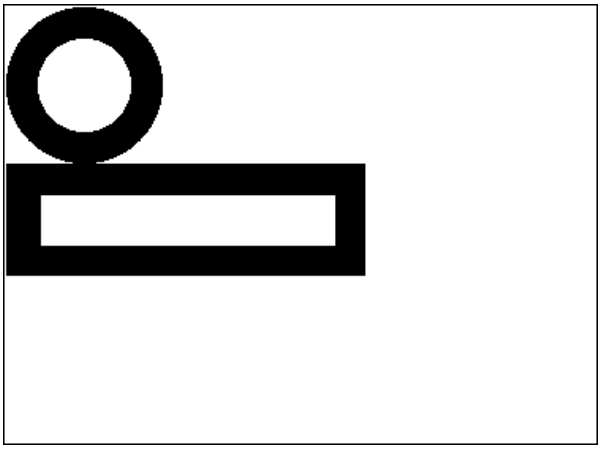


rectangle + cercle ` Placer le cercle à droite du rectangle

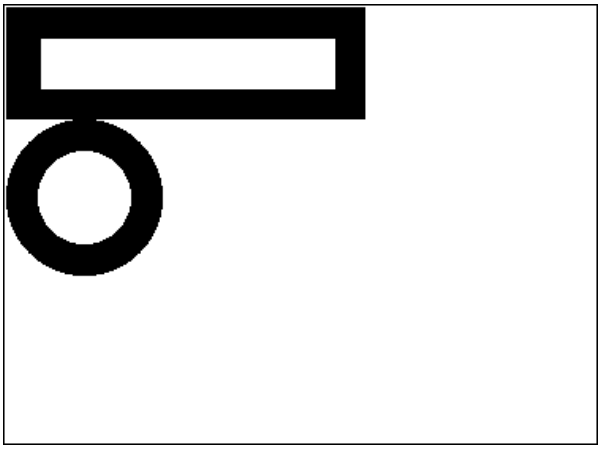


• Concaténation verticale

cercle / rectangle ` Placer le rectangle sous cercle

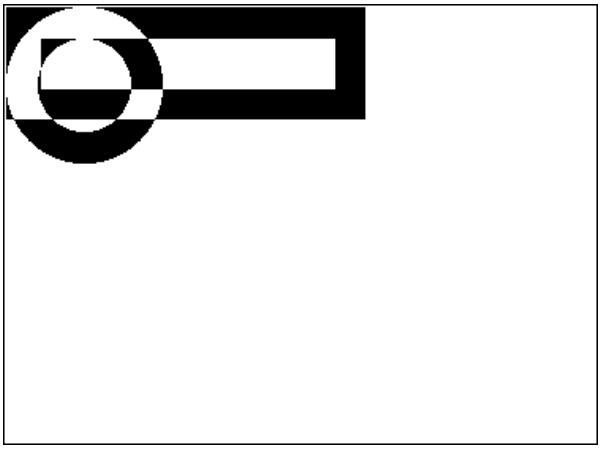


rectangle / cercle ` Placer le cercle sous le rectangle



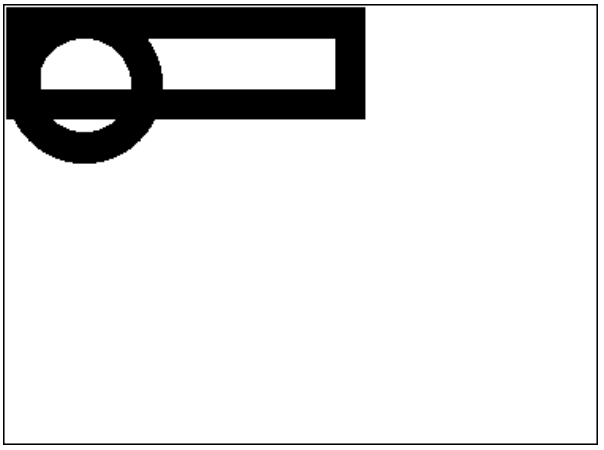
• Superposition exclusive (OU exclusif)

cercle & rectangle ` Exclusif OU des deux images



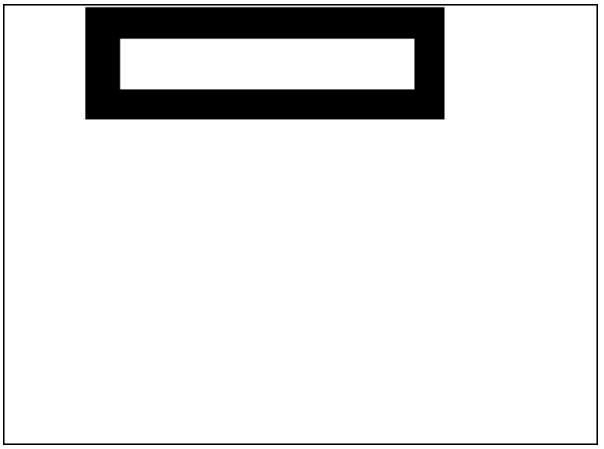
• Superposition inclusive (OU inclusif)

cercle | rectangle ` Inclusif OU des deux images

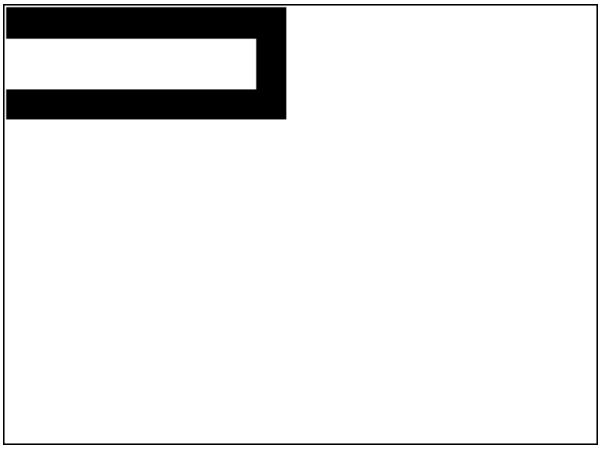


• Déplacement horizontal

rectangle + 50 ` Déplacer le rectangle 50 pixels vers la droite

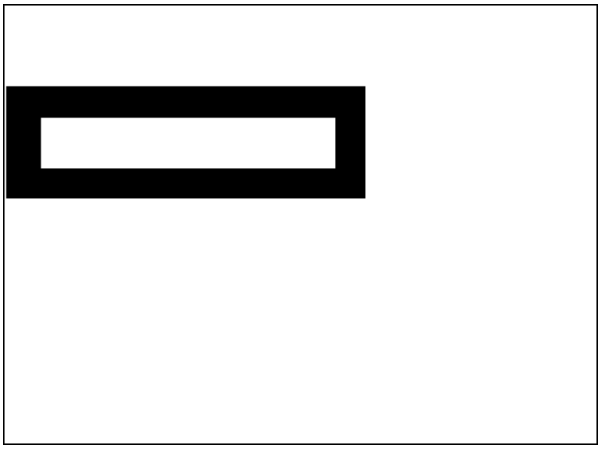


rectangle - 50 ` Déplacer le rectangle 50 pixels vers la gauche

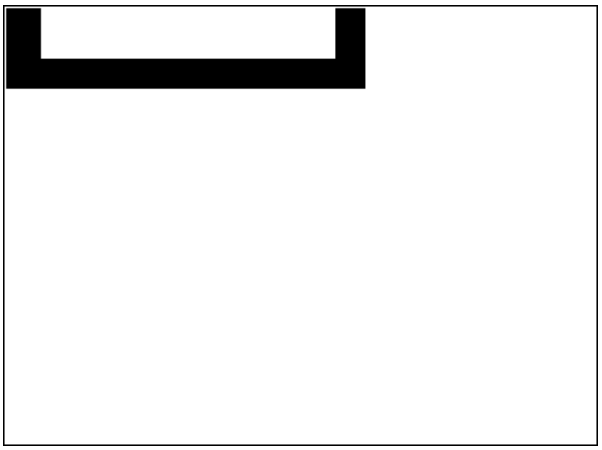


• Déplacement vertical

rectangle /50 ` Déplacer le rectangle 50 pixels vers le bas

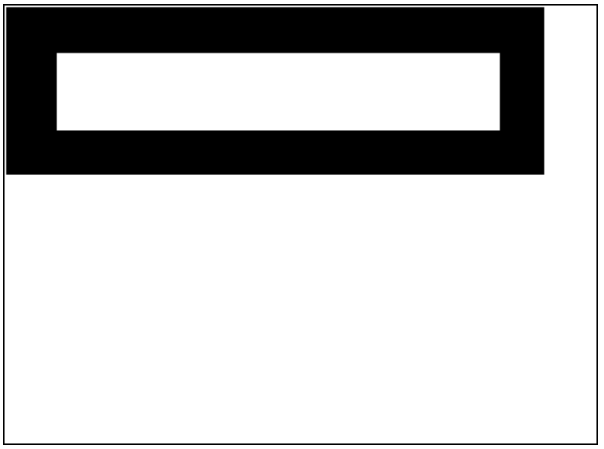


rectangle /-20 ` Déplacer le rectangle 20 pixels vers le haut

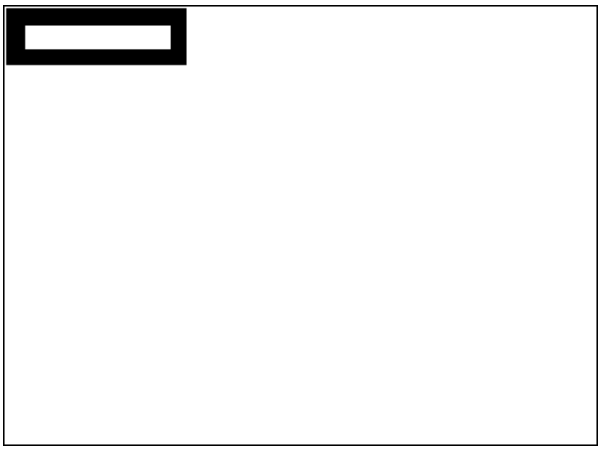


• Redimensionnement

rectangle * 1.5 ` Augmenter la taille du rectangle de 50%

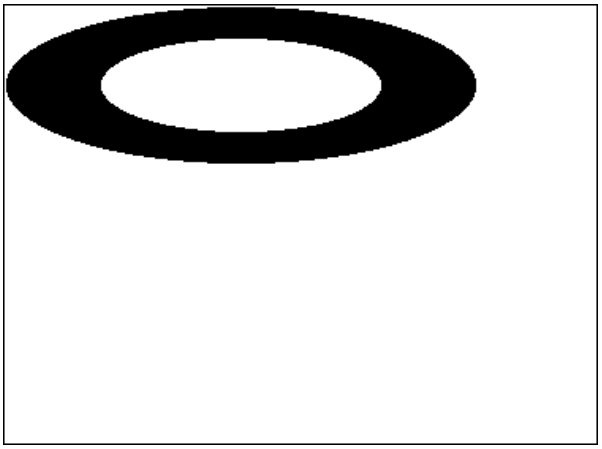


rectangle * 0.5 ` Réduire la taille du rectangle de 50%

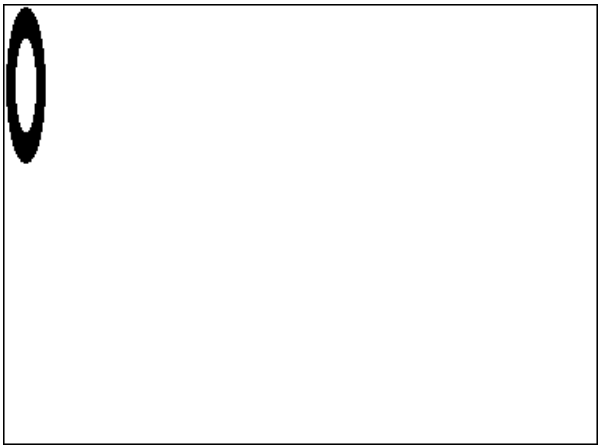


• Extension horizontale

cercle *+3 `Multiplier par 3 la largeur du cercle

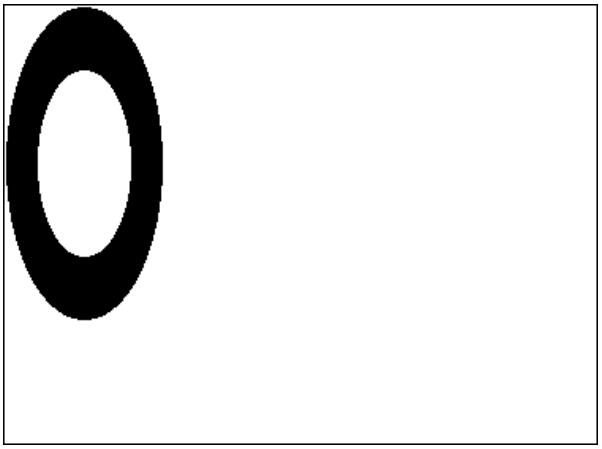


cercle *+ 0,25 ` La largeur du cercle est réduite à un quart de sa taille originale

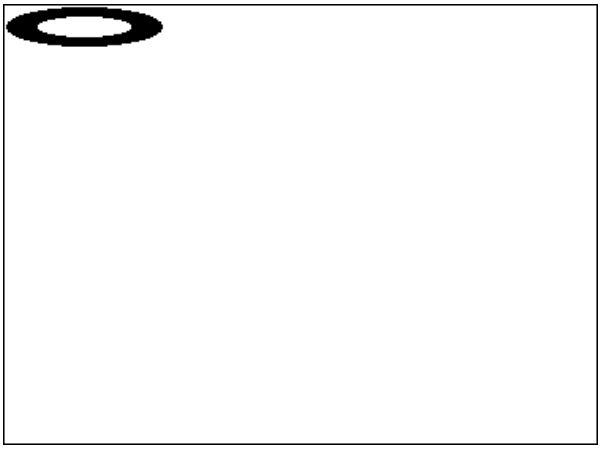


• Extension verticale

cercle */ 2 ` Doubler la hauteur du cercle



cercle */ 0.25 ` La hauteur du cercle est réduite à un quart de sa taille originale



Référence

COMBINER IMAGES, Opérateurs, Opérateurs de comparaison, Opérateurs logiques, Opérateurs numériques, Opérateurs sur les bits, Opérateurs sur les chaînes, Opérateurs sur les dates, Opérateurs sur les heures, TRANSFORMER IMAGE.

42

Outils

CHANGER DICTIONNAIRE (dictionnaire)

Paramètre	Type		Description
dictionnaire	Entier long	→	Dictionnaire à utiliser pour la correction orthographique

Description

La commande CHANGER DICTIONNAIRE provoque le remplacement du dictionnaire courant par celui spécifié par le paramètre dictionnaire. Le dictionnaire courant est utilisé pour la correction orthographique intégrée de 4D (pour plus d'informations, reportez-vous au manuel *Mode Développement*). La modification du dictionnaire courant est répercutée dans tous les process de la base pour la session.

Par défaut, 4D utilise le dictionnaire correspondant à la langue de l'application. Cinq dictionnaires principaux sont disponibles : français, anglais, allemand, espagnol et norvégien.

Passez dans le paramètre dictionnaire le numéro du dictionnaire à utiliser. Vous pouvez utiliser une des constantes prédéfinies suivantes, placées dans le thème "Dictionnaires" :

Constante	Type	Valeur
Dictionnaire anglais	Entier long	69632
Dictionnaire allemand	Entier long	131584
Dictionnaire espagnol	Entier long	196608
Dictionnaire français	Entier long	262144
Dictionnaire norvégien	Entier long	589824

En outre, de nombreuses variantes sont disponibles pour chaque langue principale. Voici la liste complète des langues et variantes prises en charge par la commande. Pour utiliser une variante, passez directement sa valeur dans le paramètre dictionnaire :

Dictionnaire	Valeur
Anglais (Royaume Uni)	65536
Anglais Irlandais (Irlande)	65600
Anglais Australien (Australie)	65664
Anglais de Nouvelle-Zélande	65680
Anglais Américain (USA)	65792
Anglais Canadien (Canada)	65920
Anglais Afrique du Sud	66048
Anglais West Indian (Caraïbes)	66176
Anglais Jamaicain (Jamaïque)	66192
Anglais (Royaume Uni + USA) 69632 (*)	

Allemand standard (Allemagne, ancienne orthographe)	131072
Allemand de Luxembourg	131073
Allemand d'Autriche	131088
Allemand du Liechtenstein	131089
Allemand de Suisse (ancienne orthographe)	131104
Allemand du Sud Tyrol	131120
Allemand nouvelle orthographe	131328
Allemand de Suisse nouvelle orthographe	131360
Allemand ancienne et nouvelle orthographe	131584 (*)
Allemand de Suisse ancienne et nouvelle orthographe	131616
Espagnol standard (Espagne)	196608 (*)
Espagnol Amérique latine standard	196864
Espagnol Argentine	196865
Espagnol Bolivie	196866
Espagnol Chili	196867
Espagnol Colombie	196868
Espagnol Cuba	196869
Espagnol Costa Rica	196870
Espagnol Rep. dominicaine	196871
Espagnol Equateur	196872
Espagnol Guatemala	196873
Espagnol Honduras	196874
Espagnol Mexique	196875
Espagnol Nicaragua	196876
Espagnol Panama	196877
Espagnol Paraguay	196878
Espagnol Perou	196879
Espagnol Puerto Rico	196880
Espagnol Salvador	196881
Espagnol Uruguay	196882
Espagnol Venezuela	196883
Espagnol Guinée Equatoriale	197121
France, Monaco, Vallée d'Aoste	262144 (*)
Français Canada	262160
Français Louisiane	262161
Français Belgique	262176
Français Luxembourg	262177
Français Suisse	262192
Français Martinique, Guadeloupe, Haïti, Guyane	262208
Français Réunion, Seychelles, Comores, Maurice	262224
Français Tahiti, Nouvelle Calédonie, Vanuatu, etc	262240
Français Maroc, Algérie, Tunisie	262256
Français Africain standard	262272
Français Bénin	262273
Français Burkina Faso	262274
Français Burundi	262275
Français Cameroun	262276
Français République Centrafricaine	262277

Français Congo (Brazzaville)	262278
Français République Démocratique du Congo (ex-Zaire)	262279
Français Côte d'Ivoire	262280
Français Djibouti	262281
Français Gabon	262282
Français Guinée	262283
Français Mauritanie	262284
Français Niger	262285
Français Rwanda	262286
Français Sénégal	262287
Français Tchad	262288
Français Togo	262289
Norvégien Bokmal	589824 (*)
Norvégien Nynorsk	590080
Norvégien Samnorsk	590336

(*) Dictionnaire principal utilisé par la constante correspondante.

Note : Le dictionnaire norvégien n'est pas présent par défaut dans les applications 4D. Veuillez contacter 4D SAS pour l'obtenir gratuitement. Vous devez ensuite l'installer dans le dossier 4D Extensions/Spellcheck.

Référence

CORRECTION ORTHOGRAPHIQUE.

Variables et ensembles système

Si le dictionnaire est correctement chargé, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est retournée.

Choisir (critère; valeur{; valeur2; ...; valeurN}) → Expression

Paramètre	Type		Description
critère	Booléen | Entier	→	Valeur à tester
valeur	Expression	→	Valeurs possibles
Résultat	Expression	←	Valeur de critère

Description

La commande Choisir retourne l'une des valeurs passées dans les paramètres valeur, valeur2, etc. en fonction de la valeur du paramètre critère.

Vous pouvez passer un paramètre critère de type booléen ou numérique :
• Si critère est un booléen, Choisir retourne valeur si le booléen vaut Vrai et valeur2 si le booléen vaut Faux. Dans ce cas, la commande attend exactement trois paramètres : critère, valeur et valeur2.
• Si critère est un entier, Choisir retourne la valeur dont la position correspond à critère. Attention, la numérotation des valeurs débute à 0 (la position de valeur est 0). Dans ce cas, la commande attend au minimum deux paramètres : critère et valeur.

La commande accepte tous les types de données pour le(s) paramètre(s) valeur, hormis les images, pointeurs, BLOBS et tableaux. Veillez cependant à ce que toutes les valeurs passées soient du même type, 4D n'effectue pas de vérification sur ce point.

Si aucune valeur ne correspond à critère, Choisir retourne une valeur "nulle" en rapport avec le type du paramètre valeur (par exemple 0 pour le type numérique, "" pour le type chaîne, etc.).

Cette commande permet de générer du code concis en remplacement des tests du type "Au cas ou" sur plusieurs lignes (cf. exemple 2). Elle est également très utile dans les emplacements où des formules peuvent être exécutées : éditeur de recherches, appliquer une formule, éditeur d'états rapides, colonne calculée de list box, etc.

Exemples

(1) Voici une utilisation type de la commande avec un critère booléen :

```
vTitre:=Choisir([Personne]Masculin;"Mr";"Madame")
```

Ce code est strictement équivalent à :

```
Si([Personne]Masculin)
    vTitre:="Mr"
Sinon
    vTitre:="Madame"
Fin de si
```

(2) Voici une utilisation type de la commande avec un critère numérique :

```
vStatut:=Choisir([Personne]Statut;"Célibataire";"Marié";"Veuf";"Divorcé")
```

Ce code est strictement équivalent à :

```
Au cas ou
    :([Personne]Statut=0)
        vStatut:="Célibataire"
    :([Personne]Statut=1)
        vStatut:="Marié"
    :([Personne]Statut=2)
        vStatut:="Veuf"
    :([Personne]Statut=3)
        vStatut:="Divorcé"
Fin de cas
```

CORRECTION ORTHOGRAPHIQUE

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande CORRECTION ORTHOGRAPHIQUE déclenche la vérification de l'orthographe du champ ou de la variable ayant le focus dans le formulaire affiché à l'écran. L'objet vérifié doit être de type Alpha ou Texte.

Note : Si vous souhaitez déclencher la correction orthographique à partir d'un bouton dans le formulaire, assurez-vous qu'il ne dispose pas de la propriété "Focusable".

La vérification débute par le premier mot du champ ou de la variable. Si un mot inconnu est détecté, la boîte de dialogue de correction apparaît (pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D). 4D utilise le dictionnaire courant (correspondant à la langue de l'application) sauf si vous avez utilisé la commande CHANGER DICTIONNAIRE.

Référence

CHANGER DICTIONNAIRE.

DECODER (blob)

Paramètre	Type		Description
blob	BLOB	→	BLOB encodé en base64
		←	BLOB décodé

Description

La commande DECODER permet de décoder le BLOB encodé en base64 passé dans le paramètre blob. La commande modifie directement le BLOB passé en paramètre.

La commande n'effectue pas de contrôle sur le contenu de blob. Vous devez veiller à ce que les données passées soient effectivement encodées en base64, sinon le résultat obtenu ne sera pas correct.

Référence

ENCODER.

ENCODER (blob)

Paramètre	Type		Description
blob	BLOB	→	BLOB à encoder en base64
		←	BLOB encodé en base64

Description

La commande ENCODER encode le BLOB passé dans le paramètre blob en base64. La commande modifie directement le BLOB passé en paramètre.

L'encodage base64 modifie des données codées sur 8 bits afin qu'elles ne conservent plus que 7 bits utiles. Cet encodage est par exemple requis pour la manipulation des BLOBs via le XML.

Référence

DECODER.

FIXER PARAMETRE MACRO

FIXER PARAMETRE MACRO (sélecteur; paramTexte)

Paramètre	Type		Description
sélecteur	Entier long	→	Sélection à utiliser
paramTexte	Texte	→	Texte envoyé

Description

La commande FIXER PARAMETRE MACRO insère le texte paramTexte dans la méthode depuis laquelle elle a été appelée.

Si du texte était sélectionné dans la méthode, le paramètre sélecteur permet de définir si le texte paramTexte doit remplacer la totalité de la méthode ou uniquement le texte sélectionné. Vous pouvez passer dans sélecteur l'une des constantes suivantes, placées dans le thème "Environnement 4D" :

Constante	Type	Valeur
Texte méthode	Entier long	1
Texte méthode surligné	Entier long	2

Si aucun texte n'était sélectionné, paramTexte est inséré dans la méthode.

Note de compatibilité : Pour que les commandes LIRE PARAMETRE MACRO et FIXER PARAMETRE MACRO fonctionnent correctement, l'attribut "version" doit être déclaré dans la macro elle-même, de la façcon suivante :
<macro name="MaMacro" version="2">
--- Texte de la macro ---
</macro>

Exemple

Cette macro construit un nouveau texte qui sera retourné à la méthode appelante

```
C_TEXTE($texte_entrée)
C_TEXTE($texte_sortie)
LIRE PARAMETRE MACRO(Texte surligné méthode;$texte_entrée)
    `Supposons que le texte sélectionné est une table, i.e. "[Clients]"
$texte_sortie:=""
    ` Tout sélectionner ([Clients])
$texte_sortie:=$texte_sortie+Nom commande(47)+"("+$texte_entrée+")"
    ` $i:=Enregistrements trouves([Clients])
$texte_sortie:=$texte_sortie+"$i:="+Nom commande(76)+"("+ $texte_entrée+")"
```

```
FIXER PARAMETRE MACRO(Texte méthode surligné;$texte_sortie)
   `On remplace le texte sélectionné par le nouveau code
```

Référence

LIRE PARAMETRE MACRO.

version 11 (Modifiée)

FIXER VARIABLE ENVIRONNEMENT (nomVar; valeurVar)

Paramètre	Type		Description
nomVar	Chaîne	→	Nom de la variable à fixer
valeurVar	Chaîne	→	Valeur de la variable ou "" pour rétablir la valeur par défaut

Description

La commande FIXER VARIABLE ENVIRONNEMENT vous permet de fixer la valeur d'une variable d'environnement sous Mac OS X et Windows. Elle est destinée à une utilisation conjointe avec les commandes FIXER EXECUTABLE CGI ou LANCER PROCESS EXTERNE. Passez dans le paramètre nomVar le nom de la variable à définir et dans le paramètre valeurVar sa valeur.

• Pour obtenir la liste générale des variables d'environnement et leurs valeurs possibles, reportez-vous à la documentation technique de votre système d'exploitation.

• Pour connaître la liste des variables d'environnement utilisables avec la commande FIXER EXECUTABLE CGI, reportez-vous à la section Support des CGI dans le chapitre "Serveur Web".

• Pour connaître la liste des variables d'environnement utilisables avec la commande LANCER PROCESS EXTERNE, reportez-vous à la documentation de cette commande. A noter que trois variables d'environnement spécifiques sont disponibles dans le cadre de l'utilisation dans ce contexte :
_4D_OPTION_CURRENT_DIRECTORY : permet de définir le répertoire courant du process externe à lancer. Vous devez passer dans valeurVar le chemin d'accès du répertoire (syntaxe type HFS sous Mac OS et DOS sous Windows).
_4D_OPTION_HIDE_CONSOLE (Windows uniquement) : permet de masquer la fenêtre de la console DOS. Vous devez passer "true" dans valeurVar pour masquer la console ou "false" pour l'afficher.
_4D_OPTION_BLOCKING_EXTERNAL_PROCESS : permet d'exécuter le process externe en mode asynchrone, c'est-à-dire non bloquant pour les autres applications. Vous devez passer "false" dans valeurVar pour définir une exécution asynchrone.
Ces variables sont valides dans le process courant pour le prochain appel à LANCER PROCESS EXTERNE.

Exemples

Reportez-vous aux exemples de la commande LANCER PROCESS EXTERNE.

Référence

FIXER EXECUTABLE CGI, LANCER PROCESS EXTERNE, Support des CGI.

LANCER PROCESS EXTERNE (nomFichier{; fluxEntrée{; fluxSortie{; fluxErreur}}})

Paramètre	Type		Description
nomFichier	Chaîne	→	Chemin d'accès et arguments du fichier à lancer
fluxEntrée	Chaîne | BLOB	→	Flux d'entrée (stdin)
fluxSortie	Chaîne | BLOB	←	Flux de sortie (stdout)
fluxErreur	Chaîne | BLOB	←	Flux d'erreur (stderr)

Attention : Cette commande est destinée aux utilisateurs avancés.

Description

La commande LANCER PROCESS EXTERNE permet de lancer un process externe depuis 4D, sous Mac OS X et Windows.
Sous Mac OS X, cette commande donne accès à toutes les applications exécutables pouvant être lancées depuis le Terminal.

Note : Pour les utilisateurs de 4D Pack, cette commande reprend et étend les possibilités offertes par la commande AP Sublaunch.

Passez dans le paramètre nomFichier le chemin d'accès absolu de l'application à exécuter ainsi que les arguments nécessaires, le cas échéant.
Sous Mac OS X, vous pouvez également passer uniquement le nom de l'application à exécuter, 4D utilisera alors la variable d'environnement PATH pour localiser l'exécutable.

Attention : Cette commande permet uniquement de lancer des applications exécutables, elle ne peut pas exécuter d'instructions dépendantes du *shell* (l'interpréteur de commandes). Par exemple, sous Mac OS il n'est pas possible d'utiliser cette commande pour exécuter l'instruction echo ou des indirections.

Le paramètre fluxEntrée (facultatif) contient le stdin du process externe. Après l'exécution de la commande, les paramètres fluxSortie et fluxErreur (s'ils sont passés) retournent respectivement le stdout et le stderr du process externe. Vous pouvez utiliser des paramètres de type BLOBs au lieu de chaînes de caractères si vous traitez des données texte de taille supérieure à 32 Ko ou des données binaires (telles que des images).

Note : Si vous utilisez la variable d'environnement _4D_OPTION_BLOCKING_EXTERNAL_PROCESS via la commande FIXER VARIABLE ENVIRONNEMENT (exécution asynchrone), les paramètres fluxSortie et fluxErreur ne sont pas retournés.

Exemples sous Mac OS X

Tous les exemples suivants utilisent le Terminal de Mac OS X, accessible dans le dossier Applications/Utilitaires.

(1) Pour modifier les accès à un fichier (chmod est la commande Mac OS X permettant de modifier les accès des fichiers) :

```
LANCER PROCESS EXTERNE ("chmod +x /dossier/monfichier.txt")
```

(2) Pour éditer un fichier texte (cat est la commande Mac OS X permettant d'éditer les fichiers). Dans cet exemple, le chemin d'accès absolu de la commande est passé :

```
C_TEXTE(vtentrée;vtsortie)
vtentrée:=""
LANCER PROCESS EXTERNE ("/bin/cat /dossier/monfichier.txt";vtentrée;vtsortie)
```

(3) Pour récupérer la liste du contenu du dossier "Users" (ls -l est semblable à la commande dir du DOS) :

```
C_TEXTE($In;$Out)
LANCER PROCESS EXTERNE("/bin/ls -l /Users";$In;$Out)
```

(4) Pour lancer une application "graphique" indépendante, il est préférable d'utiliser la commande système open (dans ce cas l'instruction LANCER PROCESS EXTERNE a le même effet qu'un double-clic sur l'application) :

```
LANCER PROCESS EXTERNE("open /Applications/Calculator.app")
```

Exemples sous Windows

(5) Pour lancer l'application NotePad :

```
LANCER PROCESS EXTERNE ("C:\\WINDOWS\\notepad.exe")
```

(6) Pour lancer l'application NotePad et ouvrir un document spécifique :

```
LANCER PROCESS EXTERNE ("C:\\WINDOWS\\notepad.exe C:\\Docs\\nouveau
                                                    dossier\\res.txt")
```

(7) Pour lancer l'application Microsoft® Word® et ouvrir un document spécifique (à noter l'emploi de deux "") :

```
$mondoc:="C:\\Program Files\\Microsoft Office\\Office10\\WINWORD.EXE
          \"C:\\Documents and Settings\\JeanMarc\\Bureau\\MesDocs\\Nouveau
                                                    dossier\\essai.xml\""
LANCER PROCESS EXTERNE($mondoc;$tIn;$tOut)
```

(8) Pour exécuter un script Perl (requiert l'installation préalable d'ActivePerl) :

```
C_TEXTE($entrée;$sortie)
FIXER VARIABLE ENVIRONNEMENT("mavariable";"valeur")
LANCER PROCESS EXTERNE ("D:\\Perl\\bin\\perl.exe
                                        D:\\Perl\\eg\\cgi\\env.pl";$entrée;$sortie)
```

(9) Pour lancer une commande avec un répertoire courant défini et sans afficher la console :

```
FIXER VARIABLE ENVIRONNEMENT("_4D_OPTION_CURRENT_DIRECTORY";
                                                    "C:\\4D_VCS")
FIXER VARIABLE ENVIRONNEMENT("_4D_OPTION_HIDE_CONSOLE";"true")
LANCER PROCESS EXTERNE("C:\\MesApplis\\macommande.exe")
```

(10) Pour permettre à l'utilisateur d'ouvrir un document externe sous Windows :

```
$nomdoc:=Selectionner document("";"*.*";"Choisissez le fichier à ouvrir";0)
Si(OK=1)
    FIXER VARIABLE ENVIRONNEMENT("_4D_OPTION_HIDE_CONSOLE";"true")
    LANCER PROCESS EXTERNE("cmd.exe /C start \"\" \""+document+"\"")
Fin de si
```

Référence

FIXER VARIABLE ENVIRONNEMENT.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système *OK* prend la valeur 1. Sinon (fichier non trouvé, mémoire insuffisante, etc.), elle prend la valeur 0.

LIRE PARAMETRE MACRO

LIRE PARAMETRE MACRO (sélecteur; paramTexte)

Paramètre	Type		Description
sélecteur	Entier long	→	Sélection à utiliser
paramTexte	Texte	←	Texte récupéré

Description

La commande LIRE PARAMETRE MACRO retourne dans paramTexte une partie ou la totalité du texte de la méthode depuis laquelle elle a été appelée.

Le paramètre sélecteur permet de définir le type d'information à récupérer. Vous pouvez passer l'une des constantes suivantes, placées dans le thème "Environnement 4D" :

Constante	Type	Valeur
Texte entier méthode	Entier long	1
Texte surligné méthode	Entier long	2

Si vous passez Texte entier méthode dans sélecteur, la totalité du texte de la méthode sera retourné dans paramTexte. Si vous passez Texte surligné méthode dans sélecteur, seul le texte sélectionné dans la méthode sera retourné dans paramTexte.

Exemple

Reportez-vous à l'exemple de FIXER PARAMETRE MACRO.

OUVRIR URL WEB (url{; *})

Paramètre	Type		Description
url	Alpha	→	URL de démarrage
*	*	→	Si spécifié = l'URL n'est pas traduit, Si omis = l'URL est traduit

Description

La commande OUVRIR URL WEB ouvre le fichier ou l'URL passé dans le paramètre url avec l'application la plus adaptée.

Le paramètre url peut contenir aussi bien un URL standard qu'un chemin d'accès de fichier.

4D tente d'abord d'interpréter le paramètre comme un chemin d'accès de fichier. La commande accepte des ':' sous Mac OS, des '\' sous Windows ou un URL posix commençant par file://. Si c'est le cas, 4D demande au système d'ouvrir le fichier avec l'application la plus adaptée (par exemple un navigateur pour les .html, Word pour les .doc, etc.). Le paramètre *, s'il est passé, est ignoré dans ce cas.

Si le paramètre url contient un URL standard (protocoles mailto:, news:, http:, etc), 4D lance le navigateur Web par défaut et accède à l'URL. S'il n'y a pas de navigateur sur les volumes connectés à l'ordinateur, la commande ne fait rien.
4D encode automatiquement les caractères spéciaux de l'URL. Si vous passez le caractère *, 4D ne traduira pas les caractères spéciaux de l'URL. Cette option permet d'accéder à ou de renvoyer des URL du type "http://www.server.net/page.htm?q=quelquechose".

Note : Cette commande ne fonctionne pas dans le cadre d'un process Web.

Exemples

(1) Les exemples suivants illustrent les différents types de chaînes acceptées comme URLs par la commande :

```
OUVRIR URL WEB( "http://www.4d.com")
OUVRIR URL WEB( "file://C:/Users/Laurent/Documents/pending.htm")
OUVRIR URL WEB( "C:\Users\Laurent\Documents\pending.htm")
OUVRIR URL WEB( "mailto:jean_martin@4d.fr")
```

(2) Cet exemple permet de lancer une application :

```
$fichier:=Selectionner document("";"";0)
Si (OK = 1)
    OUVRIR URL WEB(Document)
Fin de si
```

43

Process

Le multi-tâche dans 4D représente la possibilité d'exécuter simultanément plusieurs opérations de base de données distinctes. Ces opérations sont appelées des **process**.

Créer de multiples process équivaut à avoir plusieurs utilisateurs travaillant sur le même ordinateur, chacun effectuant sa tâche. Cela signifie principalement qu'une méthode peut être exécutée comme une tâche distincte de base de données.

Cette section traite les sujets suivants:
• Créer et effacer des process
• Eléments du process
• Process créés par 4D
• Process locaux et globaux
• Verrouillage d'enregistrements entre process.

Note : Cette section ne traite pas les procédures stockées. Pour cela, reportez-vous à la section Procédures stockées dans le manuel de référence de 4D Server.

Créer un process séparé

Il existe plusieurs manières de créer un nouveau process :
• Exécuter une méthode à partir du mode Développement en sélectionnant la case à cocher **Nouveau process** dans la boîte de dialogue d'exécution de méthode. La méthode choisie dans ce dialogue est la méthode process.
• Les process peuvent être démarrés par les commandes de menu. Dans l'éditeur de menus, sélectionnez la commande de menu et cochez l'option **Démarrer un process**. La méthode associée à la commande de menu est la méthode process.
• Utiliser la fonction Nouveau process. La méthode passée en paramètre à la fonction Nouveau process est la méthode process.
• Utiliser la fonction Executer sur serveur afin de créer une procédure stockée sur le serveur. La méthode passée en paramètre à la fonction est la méthode process.

Un process peut être supprimé dans les conditions suivantes. Les deux premières sont automatiques :
• Lorsque la méthode process a terminé son exécution,
• Lorsque l'utilisateur quitte la base,
• Si vous stoppez le process par le langage ou utilisez le bouton **Stop** dans le débogueur,
• Si vous choisissez **Tuer** dans l'Explorateur d'exécution.

Un process peut créer un autre process. Les process ne sont pas organisés hiérarchiquement. Tous les process sont égaux, et cela indépendamment du process à partir duquel ils ont été créés. Une fois créé, le process vit indépendamment de celui qui l'a créé.

Eléments d'un process

Chaque process contient certains éléments spécifiques. Il y a trois types d'éléments bien distincts dans un process :
• **Eléments d'interface** : Ce sont les éléments nécessaires à l'affichage du process.
• **Eléments de données** : Ce sont les informations liées aux données de la base.
• **Eléments de langage** : Ce sont les éléments utilisés par le langage ou importants pour le développement de l'application.

Eléments d'interface

Les éléments d'interface sont les suivants :
• **Barre de menus** : Chaque process a sa propre barre de menus courante. La barre de menus du process de premier plan est la barre de menu courante de la base.
• **Une ou plusieurs fenêtres** : Chaque process peut contrôler plusieurs fenêtres ouvertes simultanément. A l'inverse, des process peuvent n'avoir pas de fenêtre du tout.
• **Une fenêtre de premier plan** : Bien qu'un process puisse disposer de plusieurs fenêtres ouvertes simultanément, chaque process n'a qu'une fenêtre active. Pour avoir plusieurs fenêtres actives à la fois, vous devez démarrer plusieurs process.

Note : Les process exécutés sur le serveur (procédures stockées) ne doivent pas contenir d'éléments d'interface.

Eléments de données

Les éléments de données se réfèrent aux données de la base. Ce sont les suivants :
• **Sélection courante par table** : Chaque process a sa propre sélection courante. La même table peut avoir différentes sélections courantes dans différents process.
• **Enregistrement courant par table** : Chaque table peut avoir un enregistrement courant différent dans chaque process.

Note : Cette description des éléments de données est valide si les process sont des process globaux. Par défaut, tous les process sont globaux. Reportez-vous plus bas au paragraphe traitant de ce point.

Eléments de langage

Les éléments de langage d'un process sont tous les éléments liés à la programmation dans 4D.
• **Variables** : Chaque process a ses propres variables process. Reportez-vous à la section Variables pour plus d'informations. Les variables process ne sont reconnues que dans le cadre de leur process natif.

• **Table par défaut** : Chaque process a sa propre table par défaut. Cependant, notez que la commande TABLE PAR DEFAUT est seulement une convention de programmation.
• **Formulaires entrée et sortie** : Les formulaires entrée et sortie par défaut peuvent être choisis par programmation pour chaque table et chaque process.
• **LockedSet et ensembles process** : Chaque process a ses propres ensembles process. LockedSet est un ensemble process. Les ensembles process sont effacés dès que la méthode du process est terminée.
• **Appel sur erreur par process** : Chaque process a sa propre méthode de gestion d'erreurs.
• **Fenêtre de débogueur** : Chaque process peut avoir sa propre Fenêtre de débogueur.

Process utilisateur

Les process utilisateur sont des process que vous créez pour effectuer certaines tâches. Ils partagent le temps machine avec les process principaux. Les process de connexion Web sont des process utilisateur.

L'application 4D crée également des process pour ses propres besoins. Voici les principaux process créés et gérés par 4D :
• **Process principal** : Le process principal gère les fenêtres d'affichage de l'interface utilisateur.
• **Process développement** : Le process développement gère les fenêtres et éditeurs de l'environnement de Développement. Il n'y a pas de process développement dans une base compilée ne contenant pas de code interprété.
• **Process Server Web** : Le process Server Web est créé lorsque la base est publiée sur le Web. Reportez-vous à la section Mise en route du serveur Web et gestion des connexions pour plus d'informations.
• **Process Gestionnaire du cache** : Le process Gestionnaire du cache gère les entrées/sorties disque de la base. Ce process est créé dès que 4D ou 4D Server est lancé.
• **Process d'indexation** : Le process d'indexation gère les index des champs de la base dans un process séparé. Ce process est créé lorsqu'un index est créé ou détruit pour un champ.
• **Process de Gestion d'événements** : Ce process est créé quand une méthode de gestion d'événement est installée par la commande APPELER SUR EVENEMENT. Ce process exécute la méthode d'appel sur événement installée par la commande APPELER SUR EVENEMENT à chaque fois qu'un événement se produit. La méthode événement est la méthode process de ce process. Elle s'exécute continuellement, même s'il n'y a pas de méthode en exécution. La gestion d'événements fonctionne aussi en mode Développement.

Process globaux et locaux

La portée (l'aire d'action) des process peut être globale ou locale. Par défaut, tous les process sont globaux.

Dans la plupart des cas, vous utiliserez des process globaux. Les process globaux peuvent effectuer n'importe quelle opération, accéder aux données et les manipuler.

Les process locaux ne doivent être utilisés que pour des opérations qui n'accèdent pas aux données. Par exemple, vous pouvez utiliser un process local pour contrôler les éléments d'interface comme les palettes flottantes ou exécuter une méthode de gestion d'événements.

Attention : Si vous tentez d'accéder aux données à partir d'un process local, vous accédez aux données par l'intermédiaire du Process principal, et prenez donc le risque d'entrer en conflit avec les opérations effectuées dans ce process.

Vous spécifiez qu'un process est local par le biais de son nom. Le nom d'un process local doit commencer par le symbole dollar ($).

4D Server : Avec 4D Server, l'utilisation de process locaux côté client, pour des opérations qui ne nécessitent pas d'accès aux données, permet d'allouer davantage de temps machine à des tâches qui sollicitent intensivement le serveur.

Verrouillage d'enregistrements entre process

Un enregistrement est verrouillé pour un process lorsqu'un autre process l'a déjà chargé pour modification. Un enregistrement verrouillé peut être chargé par un autre process mais ne peut pas être modifié. L'enregistrement est déverrouillé seulement dans le process dans lequel l'enregistrement est modifié. Une table doit être en mode lecture/écriture pour qu'un enregistrement puisse être chargé non verrouillé. Pour plus d'informations, reportez-vous à la section Verrouillage d'enregistrements.

Référence

Méthodes, Méthodes projet, Variables.

Chercher process

version 6.0

Chercher process (nom{; *}) → Numérique

Paramètre	Type		Description
nom	Alpha	→	Nom du process duquel récupérer le numéro
*	*	→	Retourner le numéro du process serveur
Résultat	Numérique	←	Numéro du process

Description

La commande Chercher process retourne le numéro du process dont vous passez le nom dans nom. Si aucun process n'est trouvé, Chercher process retourne 0.

Le paramètre optionnel * vous permet, à partir de 4D Client, de récupérer le numéro d'un process s'exécutant sur le serveur, c'est-à-dire une procédure stockée. Dans ce cas, la valeur retournée est négative. Cette option est particulièrement utile dans le cadre de l'utilisation des commandes LIRE VARIABLE PROCESS et ECRIRE VARIABLE PROCESS. Pour plus d'informations, reportez-vous à la description de ces commandes.
Si la commande est exécutée avec le paramètre * à partir d'un process tournant sur le poste serveur, la valeur retournée est positive.

Exemple

Vous créez une palette flottante, fonctionnant dans un process séparé, dans lequel vous implémentez vos propres outils pour interagir avec l'environnement Développement. Par exemple, quand vous sélectionnez un élément dans une liste hérarchique de mots-clés, vous voulez coller du texte dans la fenêtre de premier plan du mode Développement. Pour cela, vous pouvez utiliser le presse-papiers, mais l'événement de collage doit se passer dans le process Développement. La petite fonction qui suit retourne le numéro du process de Développement (s'il est actif) :

```
` Méthode projet Numéro process Développement
` Numéro process Développement  -> Entier long
` Numéro process Développement -> Numéro du process de Développement

$0:=Chercher process("Process Développement")
   ` Note: ceci peut ne pas fonctionner si le nom du process est modifié dans l'avenir
```

Avec cette fonction, la méthode projet listée ci-dessous colle le texte reçu en paramètre dans la fenêtre de premier plan du mode Développement (si c'est possible) :

```
` Méthode projet COLLER TEXTE EN STRUCTURE
` COLLER TEXTE EN STRUCTURE ( Texte)
` COLLER TEXTE EN STRUCTURE ( Texte à coller dans la fenêtre de Structure de premier plan )

C_TEXTE($1)
C_ENTIER LONG($vlStructurePID;$vlCompte)

$vlStructurePID:=Numero process Développement
Si ($vlStructurePID # 0)
    ` Mettre le texte dans le presse-papiers
    FIXER TEXTE DANS CONTENEUR($1)
    ` Générer un événement Ctrl-V / Command-V
    GENERER FRAPPE CLAVIER(Code de caractere("v");Masque touche commande;$vlStructurePID)
    ` Appeler répétitivement ENDORMIR PROCESS pour que le minuteur puisse passer
    ` l'événement au process Développement
    Boucle ($vlCompte;1;5)
        ENDORMIR PROCESS(Numero du process courant;1)
    Fin de boucle
Fin de si
```

Référence

ECRIRE VARIABLE PROCESS, INFORMATIONS PROCESS, LIRE VARIABLE PROCESS, Statut du process.

DESINSCRIRE CLIENT

Process

version 6.5

DESINSCRIRE CLIENT

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande DESINSCRIRE CLIENT “désinscrit” le client 4D de 4D Server. Il doit avoir été préalablement inscrit à l’aide de la commande INSCRIRE CLIENT.

Si le poste client n’était pas inscrit ou si la commande est exécutée sur 4D en mode local, la commande ne fait rien.

Note : Un client 4D est automatiquement désinscrit lorsque l’application quitte.

Exemple

Reportez-vous à l’exemple de la commande INSCRIRE CLIENT.

Référence

EXECUTER SUR CLIENT, INSCRIRE CLIENT, LIRE CLIENTS INSCRITS.

Variables et ensembles système

Si le client est correctement désinscrit, la variable système *OK* prend la valeur 1. Si le client n’était pas inscrit, *OK* prend la valeur 0.

ENDORMIR PROCESS

version 5

ENDORMIR PROCESS (process; durée)

Paramètre	Type	Description
process	Numérique →	Numéro de process
durée	Numérique →	Durée exprimée en ticks

Description

ENDORMIR PROCESS permet d'endormir un process pour un certain nombre de ticks (1 tick = 1/60ème de seconde). Pendant cette période, le process endormi n'utilise pas de temps machine. Il reste cependant toujours en mémoire.

Si le process est déjà endormi, cette commande l'endort à nouveau. Le paramètre durée n'est pas ajouté au temps restant mais le remplace. Vous pouvez passer zéro (0) dans durée si vous ne voulez plus endormir le process.

Si le process n'existe pas, la commande ne fait rien.

Note : Vous ne pouvez pas utiliser cette commande à partir d'un poste client pour affecter une procédure stockée sur le poste serveur (process<0).

Exemples

(1) Reportez-vous aux exemples de la section Verrouillage d'enregistrements.
(2) Reportez-vous à l'exemple de la fonction Chercher process.

Référence

CACHER PROCESS, SUSPENDRE PROCESS.

EXECUTER SUR CLIENT

Process

version 6.5

EXECUTER SUR CLIENT (nomClient; nomMéthode{; param}{; param2; ...; paramN})

Paramètre	Type		Description
nomClient	Alpha	→	Nom d'inscription du 4D Client
nomMéthode	Alpha	→	Nom de la méthode à exécuter
param		→	Paramètre(s) de la méthode

Description

La commande EXECUTER SUR CLIENT provoque l'exécution de la méthode nomMéthode, avec, éventuellement, le(s) paramètre(s) param1... paramN, sur le ou les 4D Client inscrit(s) sous le nom nomClient. Le nom d'inscription du ou des 4D Client est défini par la commande INSCRIRE CLIENT.

Cette commande peut être appelée depuis un 4D Client ou une procédure stockée sur 4D Server.
Si la méthode admet des paramètres, passez-les après le nom de la méthode.

L'exécution de la méthode sur le 4D Client s'effectue dans un process créé automatiquement sur le poste client, et portant le nom d'inscription du 4D Client.

Si cette commande est appelée plusieurs fois de suite pour un même 4D Client, les ordres d'exécution seront empilés. Par conséquent, les méthodes seront traitées les unes à la suite des autres : les exécutions sont asynchrones. Plus l'empilement est grand, plus la "charge de travail" est grande pour le 4D Client. Vous pouvez connaître l'état de la charge de travail de chaque client à l'aide de la commande LIRE CLIENTS INSCRITS.

Note : L'empilement des ordres d'exécutions ne peut être modifié ou stoppé, sauf si le 4D Client est désinscrit à l'aide de la commande DESINSCRIRE CLIENT.

Il est possible d'exécuter simultanément la même méthode sur plusieurs ou sur la totalité des 4D Clients inscrits : pour cela, passez le caractère joker (@) dans le paramètre nomClient.

Exemples

(1) Vous souhaitez exécuter sur le poste client "Client1" la méthode "GénéreNums", comportant trois paramètres :

```
EXECUTER SUR CLIENT("Client1";"GénéreNums";12;$a;"Text")
```

(2) Vous souhaitez que tous les clients inscrits exécutent la méthode "VideTemp" :

```
EXECUTER SUR CLIENT("@";"VideTemp")
```

(3) Reportez-vous à l'exemple de la commande INSCRIRE CLIENT.

Référence

DESINSCRIRE CLIENT, INSCRIRE CLIENT, LIRE CLIENTS INSCRITS.

Variables et ensembles système

La variable système *OK* prend la valeur 1 si 4D Server a correctement reçu la requête d'exécution d'une méthode — cela ne garantit pas toutefois la bonne exécution de la méthode sur le 4D Client.

version 2004.3 (Modifiée)

Executer sur serveur (procédure; pile{; nom{; param{; param2; ...; paramN}{; *}}}) → Numérique

Paramètre	Type		Description
procédure	Alpha	→	Procédure à exécuter dans le process
pile	Numérique	→	Taille de la pile en octets
nom	Alpha	→	Nom du process créé
param	Expression	→	Paramètre(s) de la procédure
*		→	Process unique
Résultat	Numérique	←	Numéro du process pour un process nouvellement créé ou un process déjà en cours d'exécution

Description

La commande Executer sur serveur lance un nouveau process sur la machine serveur (lorsqu'elle est appelée en environnement client/serveur) et retourne le numéro de ce process.

Executer sur serveur vous permet de démarrer une procédure stockée. Pour plus d'informations sur les procédures stockées, reportez-vous à la section Procédures stockées dans le manuel de référence de 4D Server.

Si vous appelez Executer sur serveur sur un poste client, la commande retourne un numéro de process négatif. Si vous appelez Executer sur serveur sur le poste serveur, la commande retourne un numéro de process positif. A noter que l'appel de la fonction Nouveau process sur le poste serveur est équivalent à l'appel de Executer sur serveur.

Si le process n'a pas pu être créé (par exemple s'il n'y a pas assez de mémoire), Executer sur serveur retourne zéro et une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Méthode du process : Vous passez le nom de la méthode de gestion du nouveau process dans méthode. Une fois que 4D a défini le contexte pour le nouveau process, il démarre l'exécution de cette méthode qui devient alors la méthode du process.

Pile du process : Vous passez dans pile la quantité de mémoire allouée pour la pile du process. Cette valeur représente la place utilisée en mémoire pour “empiler” les appels de méthodes, les variables locales, les paramètres des sous-routines et les enregistrements empilés. Elle est exprimée en octets, il est conseillé de passer au moins 64K (environ 64000 octets) mais vous pouvez passer davantage si la chaîne d'appel dans le process (sous-routines appelant des sous-routines en cascade) est importante. Si nécessaire, vous pouvez passer par exemple 200K (environ 200000 octets).

Note : La pile n'est pas la mémoire totale réservée au process. Les process se partagent la mémoire pour les enregistrements, les variables interprocess, etc. Un process utilise également de la mémoire supplémentaire pour stocker ses variables process. La pile ne contient que les variables locales, les appels de méthodes, les paramètres des sous-routines et les enregistrements empilés.

Nom du process : Vous passez le nom du nouveau process dans nom. Avec 4D monoposte, ce nom s'affichera dans la liste des process de l'Explorateur d'exécution et sera retourné par la commande INFORMATIONS PROCESS appliquée à ce process. En client/serveur, ce nom apparaîtra en bleu dans la liste des **Procédures stockées** de la fenêtre principale de 4D Server.

Un nom de process peut contenir jusqu'à 31 caractères. Vous pouvez omettre ce paramètre ; dans ce cas, le nom du process sera une chaîne vide.

Attention : A la différence de la commande Nouveau process, vous ne devez pas avec Executer sur serveur créer un process local en préfixant son nom du symbole dollar ($). Cela fonctionnerait correctement en version monoposte, car Executer sur serveur se comporte comme Nouveau process dans cet environnement, mais, en client/serveur, cela génèrerait une erreur.

Paramètres de la méthode process : Depuis la version 6 de 4D, vous pouvez passer des paramètres à la méthode process. Vous pouvez le faire de la même manière que pour les sous-routines. Notez cependant qu'il y a une restriction : vous ne pouvez pas passer d'expression de type Pointeur. Rappelez-vous également que les tableaux ne peuvent pas être passés comme paramètres à une méthode. Une fois qu'elle a commencé à s'exécuter dans le contexte du nouveau process, la méthode process reçoit les valeurs des paramètres dans $1, $2, etc.

Note : Si vous passez des paramètres à la méthode process, vous devez passer le paramètre nom, il ne peut être omis dans ce cas.

Paramètre optionnel * : Si vous passez le dernier paramètre (optionnel) *, vous indiquez à 4D de vérifier en premier lieu si un process du même nom que celui que vous avez passé dans nom est déjà en cours d'exécution. Si c'est le cas, 4D ne démarre pas de nouveau process et retourne le numéro du process existant.

Exemple

L'exemple suivant démontre comment l'import de données peut être accéléré de manière spectaculaire en environnement client/serveur. La méthode Import classique listée ci-dessous vous permet de mesurer combien de temps prend un import d'enregistrements basé sur la commande IMPORTER TEXTE :

```
    ` Méthode projet Import classique
$vhDocRef:=Ouvrir document("")
Si (OK=1)
    FERMER DOCUMENT($vhDocRef)
    FORMULAIRE ENTREE([Table1];"Import")
    $vhStartTime:=Heure courante
    IMPORTER TEXTE([Table1];Document)
    $vhEndTime:=Heure courante
    ALERTE("L'opération a duré "+Chaine(0+($vhEndTime-$vhStartTime))+" secondes.")
Fin de si
```

Avec l'import de données classique, 4D Client analyse le fichier ASCII puis, pour chaque enregistrement, crée un nouvel enregistrement, remplit les champs avec les valeurs importées et envoie l'enregistrement au poste serveur afin qu'il soit ajouté à la base. Par conséquent, de nombreuses requêtes circulent sur le réseau. Afin d'optimiser l'opération, vous pouvez utiliser des procédures stockées pour effectuer l'import localement sur le poste serveur. Le poste client charge le document dans un BLOB et déclenche une procédure stockée en passant le BLOB comme paramètre. La procédure stockée place le BLOB dans un document sur le disque du poste serveur, puis importe le document en local. L'import des données est par conséquent effectué localement à une vitesse comparable à celle d'une version monoposte de 4D, car la plupart des requêtes transitant sur le réseau ont été éliminées. Voici la méthode projet CLIENT IMPORT. Lancée sur le poste client, elle déclenche l'exécution de la procédure stockée SERVER IMPORT, listée à la suite :

```
    ` Méthode projet CLIENT IMPORT
    ` CLIENT IMPORT ( Pointeur ; Alpha )
    ` CLIENT IMPORT ( -> [Table] ; Formulaire entrée )

C_POINTEUR($1)
C_ALPHA(31;$2)
C_HEURE($vhDocRef)
C_BLOB($vxData)
C_ENTIER LONG(spErrCode)
```

```
    ` Sélectionnez le document à importer
$vhDocRef:=Ouvrir document("")
Si (OK=1)
        ` Si un document était sélectionné, ne pas le garder ouvert
    FERMER DOCUMENT($vhDocRef)
    $vhStartTime:=Heure courante
        ` Essayons de le charger en mémoire
    DOCUMENT VERS BLOB(Document;$vxData)
    Si (OK=1)
            ` Si le document a pu être chargé dans le BLOB, démarrer
            ` la procédure stockée qui va importer les données sur le poste serveur
        $spProcessID:=Executer sur serveur("SERVER IMPORT";32*1024;
                                    "Serveur Import Services";Table($1);$2;$vxData)
            ` Nous n'avons alors plus besoin du BLOB dans ce process
        EFFACER VARIABLE($vxData)
            ` Attendons l'achèvement de l'opération effectuée par la procédure stockée
        Repeter
            ENDORMIR PROCESS(Numero du process courant;300)
            LIRE VARIABLE PROCESS($spProcessID;spErrCode;spErrCode)
            Si (Indefinie(spErrCode))
                    ` Note: si la procédure stockée n'a pas initialisé sa propre instance
                    ` de la variable spErrCode, il se peut qu'une variable indéfinie soit
                    ` retournée
                spErrCode:=1
            Fin de si
        Jusque (spErrCode<=0)
            ` Envoyons un accusé de réception à la procédure stockée
        spErrCode:=1
        ECRIRE VARIABLE PROCESS($spProcessID;spErrCode;spErrCode)
        $vhEndTime:=Heure courante
        ALERTE("L'opération a duré "+Chaine(0+($vhEndTime-$vhStartTime))+" secondes.")
    Sinon
        ALERTE("Il n'y a pas assez de mémoire pour charger le document.")
    Fin de si
Fin de si
```

Voici la méthode projet SERVER IMPORT exécutée en tant que procédure stockée :

```
    ` Méthode projet SERVER IMPORT
    ` SERVER IMPORT ( Entier long ; Alpha ; BLOB )
    ` SERVER IMPORT ( Numéro de table ; Formulaire entrée ; Données importées )

C_ENTIER LONG($1)
C_ALPHA(31;$2)
C_BLOB($3)
C_ENTIER LONG(spErrCode)

    ` L'opération n'est pas encore terminée, affectons 1 à spErrCode
spErrCode:=1
$vpTable:=Table($1)
FORMULAIRE ENTREE($vpTable->;$2)
$vsDocName:="Fichier Import "+Chaine(1+Hasard)
SUPPRIMER DOCUMENT($vsDocName)
BLOB VERS DOCUMENT($vsDocName;$3)
IMPORTER TEXTE($vpTable->;$vsDocName)
SUPPRIMER DOCUMENT($vsDocName)
    ` L'opération est terminée, affectons 0 à spErrCode
spErrCode:=0
    ` Attendons que le poste client à l'origine de la requête ait reçu les résultats
Repeter
    ENDORMIR PROCESS(Numero du process courant;1)
Jusque (spErrCode>0)
```

Une fois que ces deux méthodes projet ont été implémentées dans votre base, vous pouvez effectuer un import basé sur une procédure stockée, en écrivant par exemple :

```
CLIENT IMPORT (->[Table1];"Import")
```

Si vous réalisez quelques tests comparatifs, vous pourrez constater qu'avec ce type de méthode, l'import des enregistrements est jusqu'à 60 fois plus rapide qu'un import "classique".

Référence

Nouveau process, Procédures stockées.

INFORMATIONS PROCESS (process; procNom; procStatut; procTemps{; procVisible{; uniqueID{; origine}}})

Paramètre	Type		Description
process	Numérique	→	Numéro du process
procNom	Alpha	←	Nom du process
procStatut	Numérique	←	Statut du process
procTemps	Numérique	←	Temps d'exécution cumulé du process en ticks
procVisible	Booléen	←	Visible (Vrai) ou Caché (Faux)
uniqueID	Entier	←	Numéro unique du process
origine	Entier long	←	Origine du process

Description

La commande INFORMATIONS PROCESS retourne les informations sur le process dont vous passez le numéro dans process.

Après l'appel :
• procNom retourne le nom du process. Quelques points sont à noter à propos du nom du process :

- Si le process a été démarré depuis la boîte de dialogue **Exécuter une méthode** (avec l'option **Nouveau process** sélectionnée), son nom est “P_” suivi d'un numéro.
- Si le process a été démarré à partir d'une commande de menu personnalisé dont la propriété **Démarrer un process** est sélectionnée, le nom du process est “M_” ou “ML_” suivi d'un numéro.
- Si un process a été stoppé (et son “espace” non encore réutilisé), son nom est encore retourné. Pour détecter si un process est stoppé, testez procStatut=-1 (voir ci-dessous).

• procStatut retourne le statut du process au moment de l'appel. Ce paramètre peut retourner l'une des valeurs fournies par les constantes prédéfinies suivantes (thème "Statut du process") :

Constante	Type	Valeur
Détruit	Entier long	-1
Endormi	Entier long	1
Inexistant	Entier long	-100
En exécution	Entier long	0

Dialogue caché	Entier long	6
Suspendu	Entier long	5
En attente d'entrée sortie	Entier long	3
En attente d'un drapeau interne	Entier long	4
En attente d'événement	Entier long	2

• procTemps retourne le cumul du temps que le process a utilisé depuis qu'il a été démarré, en ticks (1/60e de seconde.)

• procVisible, s'il est spécifié, retourne VRAI si le process est visible, FAUX s'il est caché.

• uniqueID, s'il est spécifié, retourne le numéro unique du process. En effet, chaque process se voit attribuer un numéro de process ainsi qu'un numéro unique de process par session. Ce dernier permet de différencier strictement deux process ou sessions de process. Il correspond au nombre de process ayant été lancés au cours de la session de l'application 4D.

• origine, s'il est spécifié, retourne une valeur décrivant l'origine du process. Ce paramètre peut retourner l'une des valeurs fournies par les constantes prédéfinies suivantes (thème Type du process) :

Constante	**Type**	**Valeur**
Process gestionnaire de clients	Entier long	-31
Process d'activité	Entier long	-26
Process minuteur interne	Entier long	-25
Process exécution méthode SQL	Entier long	-24
Process CSM	Entier long	-22
Process de restitution	Entier long	-21
Process du fichier d'historique	Entier long	-20
Process de sauvegarde	Entier long	-19
Process 4D Server interne	Entier long	-18
Process macro éditeur de méthod	Entier long	-17
Process sur fermeture	Entier long	-16
Process interface serveur	Entier long	-15
Process exécuté sur client	Entier long	-14
Process du serveur Web	Entier long	-13
Process web 4D Client	Entier long	-12
Process web avec contexte	Entier long	-11
Autre process 4D	Entier long	-10
Tâche externe	Entier long	-9
Gestionnaire d'événement	Entier long	-8
Gestionnaire Apple Event	Entier long	-7
Gestionnaire du port série	Entier long	-6

Gestionnaire d'index	Entier long	-5
Gestionnaire du cache	Entier long	-4
Process web sans contexte	Entier long	-3
Process développement	Entier long	-2
Process principal	Entier long	-1
Aucun	Entier long	0
Process exécuté sur serveur	Entier long	1
Créé par commande de menu	Entier long	2
Créé par dialogue d'exécution	Entier long	3
Autre process utilisateur	Entier long	4

Note : Les process internes à 4D retournent une valeur négative et les process générés par l'utilisateur retournent une valeur positive.

Si le process n'existe pas, ce qui veut dire que vous n'avez pas passé un nombre inclus dans l'intervalle [1>Nombre de process], INFORMATIONS PROCESS laisse les valeurs des variables passées en paramètres inchangées.

Exemples

(1) L'exemple suivant retourne le nom, le statut, et le temps écoulé dans les variables vNom, vStatut, et vTempsPassé pour le process courant :

```
C_ALPHA(80; vNom) ` Initialiser les variables
C_ENTIER(vStatut)
C_ENTIER(vTempsPassé)
INFORMATIONS PROCESS (Numero du process courant; vNom; vStatut; vTempsPassé)
```

(2) Voir l'exemple de la section Méthode base Sur fermeture.

Référence

Nombre de process, Statut du process.

INSCRIRE CLIENT

Process

version 11.3 (Modifiée)

INSCRIRE CLIENT (nomClient{; période}{; *})

Paramètre	Type		Description
nomClient	Alpha	→	Nom de la session cliente 4D
période	Entier long	→	*** Ignoré depuis la version 11.3 ***
*	*	→	Process local

Description

La commande INSCRIRE CLIENT “inscrit” un poste client 4D sous le nom nomClient auprès de 4D Server, afin de permettre que d’autres clients ou éventuellement 4D Server (par l’intermédiaire de procédures stockées) puissent y exécuter des méthodes à l’aide de la commande EXECUTER SUR CLIENT. Une fois inscrit, un client 4D peut donc exécuter une ou plusieurs méthodes pour le compte d’autres clients.

Notes :
• Vous pouvez également inscrire automatiquement chaque poste client qui se connecte à 4D Server via l'option “Inscrire les clients au démarrage pour Exécuter sur client” dans la boîte de dialogue des Préférences (cf. manuel *Mode Développement*).
• Lorsqu’elle est utilisée avec 4D en mode local, cette commande ne fait rien.
• Plusieurs postes clients 4D peuvent avoir le même nom d’inscription.

A l’issue de l’exécution de la commande, un process, nommé nomClient, est créé sur le poste client. Ce process ne peut être détruit que par la commande DESINSCRIRE CLIENT. Si le paramètre optionnel * est passé, le process créé est local (4D ajoute automatiquement le signe $ au nom du process). Sinon, il est global.

Note de compatibilité : Depuis la version 11.3 de 4D, les mécanismes de communication serveur/client ont été optimisés. Désormais, le serveur envoie directement aux clients inscrits les requêtes d'exécution lorsque c'est nécessaire (technologie "push"). Le principe précédent selon lequel les clients interrogeaient périodiquement le serveur n'est plus utilisé. Le paramètre période est ignoré lorsqu'il est passé.

Une fois la commande exécutée, il n’est pas possible de modifier “à la volée” le nom du client 4D. Pour cela, il est nécessaire d’appeler la commande DESINSCRIRE CLIENT puis d’exécuter à nouveau INSCRIRE CLIENT.

Exemple

Les méthodes suivantes permettent de réaliser une petite messagerie entre les postes clients inscrits.

1. La méthode *INSCRIPTION* permet d'inscrire un client 4D et de le tenir prêt à recevoir un message de la part d'un autre client 4D :

```
    `Méthode INSCRIPTION
    `Il faut se désinscrire avant de s'inscrire sous un autre nom
DESINSCRIRE CLIENT
Repeter
    vNomPseudo:=Demander("Entrez votre nom :";"Utilisateur";"OK";"Annuler")
Jusque ((OK=0) | (vNomPseudo # ""))
Si (OK=0)
    ...` Ne rien faire
Sinon
    INSCRIRE CLIENT(vNomPseudo;*)
Fin de si
```

2. L'instruction suivante permet de connaître les clients inscrits. Elle peut être placée dans la Méthode base Sur ouverture :

```
    ` Méthode base Sur ouverture
PrListeClient:=Nouveau process("Liste_4DClients";32000;"Liste d'inscrits")
```

3. La méthode *Liste_4DClients* permet de récupérer tous les clients 4D inscrits et les personnes acceptant de recevoir des messages :

```
    ` Méthode Liste_4DClients
Si (Type application=4D mode distant)
        ` Le code ci-dessous n'est valable qu'en mode client-serveur
    $Ref:=Creer fenetre(100;100;300;400;-(Fenêtre palette+Avec titre de fenêtre);
                                                        "Liste d'inscrits")
    Repeter
        LIRE CLIENTS INSCRITS($ListeClient;$ListeCharge)
            `Récupération des clients inscrits dans $ListeClient
        EFFACER FENETRE($Ref)
        POSITION MESSAGE(0;0)
        Boucle ($p;1;Taille tableau($ListeClient))
            MESSAGE($ListeClient{$p}+Caractere(Retour chariot))
        Fin de boucle
            `Afficher chaque seconde
        ENDORMIR PROCESS(Numero du process courant;60)
    Jusque (Faux) ` Boucle infinie
Fin de si
```

4. La méthode *Envoyer_Message* permet d'envoyer un message à un autre client 4D inscrit.

```
    ` Méthode Envoyer_Message
$Destinataire:=Demander("Destinataire du message :";"")
    ` Saisir le nom d'une des personnes visibles dans la fenêtre générée par la
    ` méthode base Sur ouverture
Si (OK # 0)
    $Message:=Demander("Message :")   ` Contenu du message
    Si (OK # 0)
            ` Envoi du message
        EXECUTER SUR CLIENT($Destinataire;"Afficher_Message";$Message)
    Fin de si
Fin de si
```

5. La méthode *Afficher_Message* affiche le message sur le poste client :

```
    ` Méthode Afficher_Message
C_TEXTE($1)
ALERTE($1)
```

6. Enfin, cette méthode permet à un poste client de n'être plus visible par les autres clients 4D et ne plus recevoir de message :

```
    ` Méthode DÉSINSCRIPTION :
DESINSCRIRE CLIENT
```

Référence

DESINSCRIRE CLIENT, EXECUTER SUR CLIENT, LIRE CLIENTS INSCRITS.

Variables et ensembles système

Si le poste client est correctement inscrit, la variable système *OK* prend la valeur 1. Si le poste était déjà inscrit, la commande ne fait rien et *OK* prend la valeur 0.

LIRE CLIENTS INSCRITS (listeClients; nbMéthodes)

Paramètre	Type		Description
listeClients	Tab Texte	←	Liste des 4D Client enregistrés
nbMéthodes	Tab Entier long	←	Liste des méthodes restant à exécuter

Description

La commande LIRE CLIENTS INSCRITS remplit deux tableaux :

• listeClients, qui contient la liste des clients “inscrits” à l’aide de la commande INSCRIRE CLIENT.

• nbMéthodes, qui fournit liste des “charges de travail” de chaque client. La charge de travail est le nombre de méthodes qu’un 4D Client doit encore exécuter, à la demande de la commande EXECUTER SUR CLIENT.

Exemples

(1) Vous souhaitez obtenir la liste des clients inscrits et des méthodes restant à exécuter :

```
TABLEAU TEXTE($clients;0)
TABLEAU ENTIER LONG($nprocs;0)
LIRE CLIENTS INSCRITS($clients;$nprocs)
```

2) Reportez-vous à l’exemple de la commande INSCRIRE CLIENT.

Référence

DESINSCRIRE CLIENT, EXECUTER SUR CLIENT, INSCRIRE CLIENT.

Variables et ensembles système

Si l’opération se déroule correctement, la variable système *OK* prend la valeur 1.

Nombre de process

version 5

Nombre de process → Entier

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier	←	Nombre total de process ouverts (y compris les process du moteur de 4D)

Description

Nombre de process retourne le nombre de process ouverts sur un poste 4D Client, 4D Server (procédures stockées) ou dans une version monoposte de 4D.

Ce nombre inclut tous les process, qu'ils soient créés par vos soins ou par 4D. Le Process principal, le Gestionnaire de cache, le Process développement, le Gestionnaire d'index ou le Process du serveur Web par exemple sont des process créés automatiquement par 4D.

La valeur retournée par Nombre de process comprend également les process qui ont été détruits.

Exemples

Référez-vous à l'exemple de Statut du process et Méthode base Sur fermeture.

Référence

INFORMATIONS PROCESS, Nombre de process utilisateurs, Nombre utilisateurs, Statut du process.

Nombre de process utilisateurs

version 11.3 (Modifiée)

Nombre de process utilisateurs → Entier

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier	←	Nombre de process vivants (à l'exception de process internes)

Description

Nombre de process utilisateurs retourne le nombre courant de process "vivants" dans l'application 4D et dont le type est différent de -25 (Process minuteur interne), -31 (Process gestionnaire de clients) et -15 (Process interface serveur). Pour plus d'informations sur les types de process, reportez-vous à la commande INFORMATIONS PROCESS et au thème de constantes Type du process.

Note : Les process "vivants" sont des process dont le statut n'est ni détruit, ni inexistant (cf. commande Statut du process).

Référence

Nombre de process, Nombre utilisateurs.

Nombre utilisateurs

Process

version 5

Nombre utilisateurs → Entier

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier	←	Nombre d'utilisateurs connectés au serveur

Description

Lorsqu'elle est appelée depuis une procédure stockée sur le serveur, la commande Nombre utilisateurs retourne le nombre d'utilisateurs connectés au poste serveur.

Si le serveur exécute au moins une procédure stockée et si Nombre utilisateurs est appelée depuis un autre contexte (poste client, méthode Web), la commande retourne le nombre d'utilisateurs +1.

Dans le cas d'une version monoposte de 4D, Nombre utilisateurs retourne 1.

Référence

Nombre de process, Nombre de process utilisateurs.

Nouveau process

version 2004.3 (Modifiée)

Nouveau process (méthode; pile{; nom{; param{; param2; ...; paramN}{; *}}}) → Numérique

Paramètre	Type		Description
méthode	Alpha	→	Méthode à exécuter dans le process
pile	Numérique	→	Taille de la pile en octets
nom	Alpha	→	Nom du process créé
param	Expression	→	Paramètre(s) de la méthode
*		→	Process unique
Résultat	Numérique	←	Numéro du process nouvellement créé ou du process déjà en cours d'exécution

Description

La commande Nouveau process lance un nouveau process (sur la même machine) et retourne le numéro de ce process.

Si le process n'a pas pu être créé, par exemple s'il n'y a pas assez de mémoire, Nouveau process retourne zéro et une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Méthode du process : Vous passez le nom de la méthode de gestion du nouveau process dans méthode. Une fois que 4D a défini le contexte pour le nouveau process, il démarre l'exécution de cette méthode qui devient alors la méthode du process.

Pile du process : Vous passez dans pile la quantité de mémoire allouée pour la pile du process. Cette valeur représente la place utilisée en mémoire pour “empiler” les appels de méthode, les variables locales, les paramètres des sous-routines et les enregistrements empilés. Elle est exprimée en octets, il est conseillé de passer au moins 64K (environ 64 000 octets) mais vous pouvez passer davantage si la chaîne d'appel dans le process (sous-routines appelant des sous-routines en cascade) est importante. Si nécessaire, vous pouvez passer par exemple 200K (environ 200000 octets).

Note : La pile n'est pas la mémoire totale réservée au process. Les process se partagent la mémoire pour les enregistrements, les variables interprocess, etc. Un process utilise également de la mémoire supplémentaire pour stocker ses variables process. La pile contient diverses informations internes à 4D ; la taille de ces informations varie en fonction du nombre d'appels de méthodes imbriquées.

Nom du process : Vous passez le nom du nouveau process dans nom. Ce nom s'affichera dans la liste des process de l'Explorateur d'exécution et sera retourné par la commande INFORMATIONS PROCESS. Un nom de process peut contenir jusqu'à 31 caractères. Vous pouvez omettre ce paramètre ; dans ce cas, le nom du process sera une chaîne vide. Vous pouvez créer un process local en préfixant son nom d'un symbole dollar ($).

Important : Rappelez-vous que, en client/serveur, les process locaux ne doivent pas accéder aux données.

Paramètres de la méthode process : Depuis la version 6 de 4D, vous pouvez passer des paramètres à la méthode process. Vous pouvez le faire de la même manière que pour les sous-routines. Notez cependant qu'il y a une restriction : vous ne pouvez pas passer d'expression de type Pointeur. Rappelez-vous également que les tableaux ne peuvent pas être passés comme paramètres à une méthode. Une fois qu'elle a commencé à s'exécuter dans le contexte du nouveau process, la méthode process reçoit les valeurs des paramètres dans $1, $2, etc.

Note : Si vous passez des paramètres à la méthode process, vous devez passer le paramètre nom, il ne peut être omis dans ce cas.

Si vous passez le dernier paramètre (optionnel) *, vous indiquez à 4D de vérifier en premier lieu si un process du même nom que celui que vous avez passé dans nom est déjà en cours d'exécution. Si c'est le cas, 4D ne démarre pas de nouveau process et retourne le numéro du process existant.

Exemple

Examinons la méthode projet suivante :

```
     ` AJOUT CLIENTS
FIXER BARRE MENUS (1)
Repeter
   AJOUTER ENREGISTREMENT([Clients];*)
Jusque (OK=0)
```

Si vous associez cette méthode projet à une commande de menu créé dans l'éditeur de barres de menus et que vous lui affectez la propriété **Démarrer un process**, 4D va automatiquement créer un nouveau process lors de l'exécution de la méthode. L'instruction CHANGER BARRE(1) associe cette barre de menus au nouveau process. En l'absence de toute fenêtre (que vous pourriez avoir ouverte avec Creer fenetre), l'appel à AJOUTER ENREGISTREMENT en créera une automatiquement.

Si maintenant vous voulez pouvoir démarrer le process Ajout Clients lorsque vous cliquez sur un bouton situé dans un tableau de contrôle personnalisé, vous pouvez écrire :

```
    ` Méthode objet bouton bAjoutClients
$vlProcessID:=Nouveau process("Ajout Clients";32*1024;"Ajout de clients")
```

Ce bouton fait la même chose que la commande de menu personnalisée.

Si, maintenant, lorsque la commande de menu est sélectionnée ou lorsque le bouton reçoit un clic, vous voulez que le process soit lancé s'il n'existe pas ou qu'il soit passé au premier plan s'il existe déjà, vous pouvez créer la méthode DEMARRER AJOUT CLIENTS :

```
    ` DEMARRER AJOUT CLIENTS
$vlProcessID:=Nouveau process("Ajout Clients";64*1024;"Ajout de clients ";*)
Si ($vlProcessID#0)
    PASSER AU PREMIER PLAN ($vlProcessID)
Fin de si
```

La méthode objet de bAjoutClient devient :

```
    ` Méthode objet bouton bAjoutClients
DEMARRER AJOUT CLIENTS
```

Dans l'éditeur de barres de menus, vous remplacez AJOUT CLIENTS par la méthode DEMARRER AJOUT CLIENTS. Désélectionnez l'option **Démarrer un process** pour la commande de menu.

Référence

Executer sur serveur, Introduction aux process, Méthodes, Méthodes projet, Variables.

Numero du process courant

version 5

Numero du process courant → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Numéro du process en cours d'exécution

Description

Numero du process courant retourne le numéro du process à partir duquel la fonction a été appelée.

Exemples

Référez-vous aux exemples de ENDORMIR PROCESS et INFORMATIONS PROCESS.

Référence

Chercher process, INFORMATIONS PROCESS, Statut du process.

Process interrompu

version 6.5

Process interrompu → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai = le process est sur le point d'être interrompu, Faux = le process n'est pas sur le point d'être interrompu

Description

La commande Process interrompu retourne Vrai si le process dans lequel elle est appelée est sur le point d'être interrompu de manière inopinée — c'est-à-dire sans être parvenu au terme "normal" de son exécution. Cela peut se produire, par exemple, à la suite d'un appel à QUITTER 4D.

Exemple

Cette commande peut être utilisée dans le cadre d'un type particulier de programmation du serveur Web, en mode compilé uniquement : lorsque vous utilisez une méthode envoyant des pages Web à l'aide d'une boucle du type Tant que...Fin tant que, le mécanisme du serveur Web ne permet pas de stopper la boucle en cas de timeout (fin de période d'inactivité autorisée) avec un browser. Le process Web n'étant pas refermé, des ressources sont ainsi gaspillées.
La commande Process interrompu, placée dans le test initial de la boucle, retournera Vrai en cas de timeout. La boucle pourra donc être interrompue et le process tué.
Voici une méthode parfois utilisée pour envoyer des pages HTML. En mode compilé, cette boucle ne pourra pas être interrompue en cas de timeout :

```
Tant que (Vrai)
   ENVOYER FICHIER HTML (FichierHTML)
Fin tant que
```

La commande Process interrompu permet d'utiliser le même type de méthode, tout en préservant la possibilité de sortir de la boucle en cas de timeout :

```
Tant que (Non (Process interrompu))
   ENVOYER FICHIER HTML (FichierHTML)
Fin tant que
```

REACTIVER PROCESS

REACTIVER PROCESS (process)

Paramètre	Type		Description
process	Numérique	→	Numéro de process

Description

REACTIVER PROCESS réactive un process suspendu ou endormi. Si process n'est pas endormi ou suspendu, REACTIVER PROCESS ne fait rien.

Si process a été suspendu, référez-vous aux commandes SUSPENDRE PROCESS ou ENDORMIR PROCESS. Si process n'existe pas, cette commande ne fait rien.

Note : Vous ne pouvez pas utiliser cette commande à partir d'un poste client pour affecter une procédure stockée sur le poste serveur (process<0).

Référence

ENDORMIR PROCESS, SUSPENDRE PROCESS.

Statut du process

Statut du process (process) → Numérique

Paramètre	Type		Description
process	Numérique	→	Numéro du process
Résultat	Numérique	←	Statut du process

Description

La commande Statut du process retourne le statut du process dont le numéro est passé dans process.

Le résultat de la fonction peut être l'une des valeurs des constantes prédéfinies suivantes :

Constante	Type	Valeur
Détruit	Entier long	-1
Endormi	Entier long	1
Inexistant	Entier long	-100
En exécution	Entier long	0
Dialogue caché	Entier long	6
Suspendu	Entier long	5
En attente entrée sortie	Entier long	3
En attente drapeau interne	Entier long	4
En attente événement	Entier long	2

Si le process n'existe pas (ce qui signifie le numéro que vous avez passé est hors de l'intervalle de 1 à Nombre de process), Statut du process retourne Inexistant (-100).

Exemple

L'exemple suivant retourne le nom et le numéro de référence du process de chaque process dans les tableaux asProcName et aiProcNum. La méthode teste si le process a été détruit. Dans ce cas, le nom et le numéro du process ne sont pas ajoutés dans le tableau :

```
$vlNbTasks:=Nombre de process
TABLEAU ALPHA(31;asProcName; $vlNbTasks)
TABLEAU ENTIER(aiProcNum; $vlNbTasks)
$vlActualCount:=0
```

```
Boucle ($vlProcess;1; $vlNbTasks)
    Si (Statut du process($vlProcess)>=En exécution)
        $vlActualCount:=$vlActualCount+1
        INFORMATIONS PROCESS($vlProcess; asProcName{$vlActualCount};$vlState;
                                                                    $vlTime)
        aiProcNum{$vlActualCount}:=$vlProcess
    Fin de si
Fin de boucle
    ` Eliminer les éléments superflus
TABLEAU ALPHA(31;asProcName;$vlActualCount)
TABLEAU ENTIER(aiProcNum;$vlActualCount)
```

Référence

INFORMATIONS PROCESS, Nombre de process.

SUSPENDRE PROCESS

SUSPENDRE PROCESS (process)

Paramètre	Type	Description
process	Numérique →	Numéro de process

Description

SUSPENDRE PROCESS suspend l'exécution de process jusqu'à ce qu'il soit remis en action par la comande REACTIVER PROCESS. Pendant ce temps, process n'utilise pas de temps machine. Lorsqu'un process est suspendu, il existe toujours en mémoire.

Si process est déjà suspendu, SUSPENDRE PROCESS ne fait rien. Si le process est endormi à l'aide de ENDORMIR PROCESS, le process est suspendu. S'il reçoit l'ordre REACTIVER PROCESS, le process redevient actif immédiatement.

Lorsqu'un process est suspendu, les fenêtres qui lui appartiennent ne sont pas saisissables. Dans ce cas, si vous ne voulez pas dérouter l'utilisateur, il faut auparavant cacher le process. Si process n'existe pas, cette commande ne fait rien.

Attention : Utilisez SUSPENDRE PROCESS seulement avec les process que vous avez créés. SUSPENDRE PROCESS n'a aucun effet sur le process principal.

Note : Vous ne pouvez pas utiliser cette commande à partir d'un poste client pour affecter une procédure stockée sur le poste serveur (process<0).

Référence

CACHER PROCESS, ENDORMIR PROCESS, REACTIVER PROCESS.

44

Process (Communications)

APPELER PROCESS (process)

Paramètre	Type	Description
process	Numérique →	Numéro du process

Description

APPELER PROCESS appelle le formulaire affiché dans la fenêtre au premier plan de process.

Important : APPELER PROCESS ne fonctionne qu'avec des process tournant sur la même machine.

Si vous appelez un process qui n'existe pas, la commande ne fait rien.

Si process (le process appelé) n'a aucune fenêtre ou si aucun formulaire n'est affiché, rien ne se passe. Le formulaire affiché dans le process appelé reçoit un événement Sur appel extérieur. Cet événement doit avoir été sélectionné pour le formulaire dans la fenêtre des **propriétés de formulaire** en mode Développement, et vous devez le traiter dans la méthode formulaire. Si l'événement n'est pas sélectionné ou géré dans la méthode formulaire, la commande ne fait rien.

Note : La réception de l'événement Sur appel extérieur dans un formulaire entrée provoque le changement du contexte de saisie du formulaire. En particulier, si un champ était en cours de modification, l'événement formulaire Sur données modifiées est généré.

Le process appelant (dans lequel la commande APPELER PROCESS est exécutée) n'attend pas : APPELER PROCESS a un effet immédiat. Il est de votre ressort d'écrire, si nécessaire, une boucle d'attente pour traiter une éventuelle réponse du process appelé à l'aide des variables interprocess ou des variables process (réservées à cette utilisation) pouvant être lues et écrites entre les deux process avec les commandes LIRE VARIABLE PROCESS et ECRIRE VARIABLE PROCESS.

Si vous voulez établir une communication entre des process qui n'affichent pas de formulaires, utilisez les commandes LIRE VARIABLE PROCESS et ECRIRE VARIABLE PROCESS.

APPELER PROCESS accepte la syntaxe alternative APPELER PROCESS (-1).
Pour ne pas ralentir l'exécution d'une méthode, 4D ne redessine pas les variables interprocess à chaque fois qu'elles sont modifiées. Si vous passez -1 au lieu du numéro du process dans le paramètre process de la commande APPELER PROCESS, toutes les variables interprocess affichées dans toutes les fenêtres de tous les process seront mises à jour et redessinées.

Exemple

Reportez-vous à l'exemple de la section Méthode base Sur fermeture.

Référence

ECRIRE VARIABLE PROCESS, Evenement formulaire, LIRE VARIABLE PROCESS.

ECRIRE VARIABLE PROCESS

version 6.0

ECRIRE VARIABLE PROCESS (process; varDestination; exprSource{; varDestination2; exprSource2; ...; varDestinationN; exprSourceN})

Paramètre	Type		Description
process	Numérique	→	Numéro de process de destination
varDestination	Variable	→	Variable de destination
exprSource	Variable	→	Expression source (ou variable source)

Description

La commande ECRIRE VARIABLE PROCESS écrit la ou les valeur(s) de exprSource (exprSource2, etc.) dans la ou les variable(s) process varDestination (varDestination2, etc.) du process de destination dont le numéro est passé dans process.

Chaque variable de destination peut être une variable ou un élément de tableau. Tenez cependant compte des restrictions évoquées ci-dessous.

Pour chaque association varDestination;exprSource, le type de l'expression doit être compatible avec la variable de destination, sinon vous pourrez obtenir des variables avec des valeurs incorrectes. En mode interprété, si la variable de destination n'existe pas, elle est créée et reçoit l'expression. En mode compilé, si aucune variable n'est associée au process de destination, une erreur est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Lorsque le process courant écrit les variables du process de destination, ce dernier n'est averti en aucune manière de l'écriture de l'instance de ses variables par un autre process.

4D Server : A partir d'un 4D Client, vous pouvez écrire des variables dans un process de destination exécuté sur le poste serveur (procédure stockée). Pour cela, passez dans process le numéro du process serveur en négatif, c'est-à-dire précédé du signe - (moins).
Attention, la communication process “intermachine” permise par les commandes ECRIRE VARIABLE PROCESS, LIRE VARIABLE PROCESS et VARIABLE VERS VARIABLE n’est possible que du client vers le serveur. C’est toujours un process client qui lit ou écrit les variables d’une procédure stockée.

Astuce : Si vous ne connaissez pas le numéro du process serveur de destination, vous pouvez tout de même écrire dans les variables interprocess du serveur. Pour cela, il vous suffit de passer toute valeur négative dans process. En d'autres termes, il n'est pas nécessaire de connaître précisément le numéro d'un process exécuté sur le serveur pour utiliser ECRIRE VARIABLE PROCESS avec des variables interprocess du serveur.
Cette possibilité s'avère particulièrement utile dans le cas d'une procédure stockée lancée sur le serveur par l'intermédiaire de la Méthode base Sur démarrage serveur. Comme les postes clients ne connaissent pas automatiquement le numéro de ce process serveur, il vous suffit de passer une valeur négative (n'importe laquelle) dans le paramètre process.

Restrictions
ECRIRE VARIABLE PROCESS n'accepte pas de variables locales comme variables de destination.

ECRIRE VARIABLE PROCESS accepte tout type de variable process ou interprocess de destination, à l'exception :
• des variables de type Pointeur.
• des tableaux de tous types. Pour écrire un tableau entier d'un process vers un autre, utilisez la commande VARIABLE VERS VARIABLE. Notez cependant que ECRIRE VARIABLE PROCESS vous permet d'écrire des éléments de tableaux.
• des éléments de tableaux de pointeurs et des éléments de tableaux à deux dimensions.

Le process de destination doit être un process utilisateur, ce ne peut être un des process du moteur de 4D. Si le process de destination n'existe pas, la commande ne fait rien.

Exemples
(1) La ligne de code suivante affecte une chaîne vide à la variable Texte vtCurStatus du process dont le numéro est $vlProcess :

```
ECRIRE VARIABLE PROCESS($vlProcess;vtCurStatus;"")
```

(2) La ligne de code suivante affecte la variable Texte vtCurStatus du process dont le numéro est $vlProcess à la valeur de la variable $vtInfo depuis la méthode en cours d'exécution du process courant :

```
ECRIRE VARIABLE PROCESS($vlProcess;vtCurStatus;$vtInfo)
```

(3) La ligne de code suivante affecte la variable Texte vtCurStatus du process dont le numéro est $vlProcess à la valeur de la même variable dans le process courant :

```
ECRIRE VARIABLE PROCESS($vlProcess;vtCurStatus;vtCurStatus)
```

Note : La première vtCurStatus désigne l'instance de la variable dans le process de destination, la seconde vtCurStatus désigne l'instance de la variable dans le process courant.

(4) L'exemple suivant place séquentiellement en majuscules les éléments d'un tableau process depuis le process désigné par $vlProcess:

```
LIRE VARIABLE PROCESS($vlProcess;vl_IPCom_Array;$vlSize)
Boucle($vlElem;1;$vlSize)
    LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
    ECRIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array{$vlElem};Majusc($vtElem))
Fin de boucle
```

Note : Dans cet exemple, la variable process vl_IPCom_Array doit être gérée par les process source/destination et contient la taille du tableau at_IPCom_Array.

(5) L'exemple suivant écrit l'instance des variables v1, v2, v3 dans le process de destination à partir de l'instance de ces mêmes variables dans le process courant :

```
ECRIRE VARIABLE PROCESS($vlProcess;v1;v1;v2;v2;v3;v3)
```

Référence

APPELER PROCESS, Introduction aux process, LIRE VARIABLE PROCESS, VARIABLE VERS VARIABLE.

EFFACER SEMAPHORE (sémaphore)

Paramètre	Type		Description
sémaphore	Alpha	→	Sémaphore à effacer

Description

EFFACER SEMAPHORE permet d'effacer le sémaphore précédemment créé par la fonction Semaphore.

La règle d'utilisation est que tous les sémaphores doivent être effacés lorsqu'ils ne sont plus nécessaires. Si les sémaphores ne sont pas effacés, ils restent en mémoire jusqu'à la fermeture du process dans lequel ils ont été créés.
Un process ne peut effacer que les sémaphores qu'il a créés. Si vous tentez d'effacer un sémaphore depuis un autre process que celui qui l'a créé, EFFACER SEMAPHORE ne fait rien.

Note : Attention, 4D tient compte de la casse des caractères en ce qui concerne les noms de sémaphores (le programme considère par exemple que "MonSémaphore" est différent de "monsémaphore").

Exemple

Reportez-vous à l'exemple de la fonction Semaphore.

Référence

Semaphore.

LIRE VARIABLE PROCESS (process; varSource; varDestination{; varSource2; varDestination2; ...; varSourceN; varDestinationN})

Paramètre	Type		Description
process	Numérique	→	Numéro de process source
varSource	Variable	→	Variable source
varDestination	Variable	←	Variable de destination

Description

La commande LIRE VARIABLE PROCESS lit la valeur de la ou des variable(s) process varSource (varSource2, etc.) depuis le process source dont le numéro est passé dans process et la retourne dans la ou les variables(s) varDestination (varDestination2, etc.) du process courant.

Chaque variable source peut être une variable, un tableau ou un élément de tableau. Tenez cependant compte des restrictions évoquées plus bas.

Pour chaque association varSource;varDestination les types des deux variables doivent être compatibles, sinon vous pourrez obtenir des valeurs non significatives.

Le process courant "pille" les variables du process de destination : ce dernier n'est averti en aucune manière de la lecture de l'instance de ses variables par un autre process.

4D Server : A partir d'un 4D Client, vous pouvez lire des variables dans un process de destination exécuté sur le poste serveur (procédure stockée). Pour cela, passez dans process le numéro du process serveur en négatif, c'est-à-dire précédé du signe - (moins).
Attention, la communication process “intermachine” permise par les commandes LIRE VARIABLE PROCESS, ECRIRE VARIABLE PROCESS et VARIABLE VERS VARIABLE n’est possible que du client vers le serveur. C’est toujours un process client qui lit ou écrit les variables d’une procédure stockée.

Astuce : Si vous ne connaissez pas le numéro du process serveur source, vous pouvez tout de même lire les variables interprocess du serveur. Pour cela, il vous suffit de passer toute valeur négative dans process. En d'autres termes, il n'est pas nécessaire de connaître précisément le numéro d'un process exécuté sur le serveur pour utiliser LIRE VARIABLE PROCESS avec les variables interprocess du serveur.

Cette possibilité s'avère particulièrement utile dans le cas d'une procédure stockée lancée sur le serveur par l'intermédiaire de la Méthode base Sur démarrage serveur. Comme, par défaut, les postes clients ne connaissent pas le numéro de ce process serveur, il vous suffit de passer une valeur négative (n'importe laquelle) dans le paramètre process.

Restrictions

LIRE VARIABLE PROCESS n'accepte pas de variables locales comme variables sources.
En revanche, les variables de destination peuvent être interprocess, process ou locales. Vous pouvez "recevoir" les valeurs uniquement dans des variables, pas dans des champs.

LIRE VARIABLE PROCESS accepte tout type de variable source, process ou interprocess, à l'exception des variables de type :
• Pointeur
• Tableau de pointeurs
• Tableau à deux dimensions

Le process source doit être un process utilisateur, ce ne peut être un des process du moteur de 4D. Si le process source n'existe pas, la commande ne fait rien.

Note : En mode interprété, si une variable source n'existe pas, la valeur indéfinie est retournée. Vous pouvez le détecter en testant la variable de destination correspondante à l'aide de la fonction Type. En mode compilé, si aucune variable n'est associée au process source, une erreur est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Exemples

(1) La ligne de code suivante lit la valeur de la variable Texte vtCurStatus dans le process dont le numéro est $vlProcess et retourne le résultat dans la variable process vtInfo du process courant :

```
LIRE VARIABLE PROCESS($vlProcess;vtCurStatus;vtInfo)
```

(2) La ligne de code suivante fait la même chose mais retourne la valeur dans la variable locale $vtInfo de la méthode s'exécutant dans le process courant :

```
LIRE VARIABLE PROCESS($vlProcess;vtCurStatus;$vtInfo)
```

(3) La ligne de code suivante fait la même chose mais retourne la valeur dans la même variable vtCurStatus du process courant :

```
LIRE VARIABLE PROCESS($vlProcess;vtCurStatus;vtCurStatus)
```

Note : La première vtCurStatus désigne l'instance de la variable dans le process source, la seconde vtCurStatus désigne l'instance de la variable dans le process courant.

(4) L'exemple suivant lit séquentiellement les éléments d'un tableau process depuis le process indiqué par $vlProcess :

```
LIRE VARIABLE PROCESS($vlProcess;vl_IPCom_Array;$vlSize)
Boucle($vlElem;1;$vlSize)
    LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
        ` Faire quelque chose avec $vtElem
Fin de boucle
```

Note : Dans cet exemple, la variable process vl_IPCom_Array doit être gérée par le process source et contient la taille du tableau at_IPCom_Array.

(5) L'exemple suivant fait la même chose que le précédent mais lit le tableau dans son intégralité au lieu de le faire élément par élément :

```
LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Array;$anArray)
Boucle($vlElem;1;Taille tableau($anArray))
    ` Faire quelque chose avec $anArray{$vlElem}
Fin de boucle
```

(6) L'exemple suivant lit l'instance des variables v1,v2,v3 dans le process source et retourne leurs valeurs dans l'instance des mêmes variables du process courant :

```
LIRE VARIABLE PROCESS($vlProcess;v1;v1;v2;v2;v3;v3)
```

(7) Reportez-vous à l'exemple de la commande PROPRIETES GLISSER DEPOSER.

Référence

APPELER PROCESS, ECRIRE VARIABLE PROCESS, Introduction aux process, Présentation du Glisser-Déposer, PROPRIETES GLISSER DEPOSER, VARIABLE VERS VARIABLE.

Semaphore (sémaphore{; nbTicks}) → Booléen

Paramètre	Type		Description
sémaphore	Alpha	→	Sémaphore à tester et à positionner
nbTicks	Entier	→	Temps d'attente maximum
Résultat	Booléen	←	sémaphore a été correctement créé (Faux) ou sémaphore était déjà créé (Vrai)

Description

Un sémaphore est un drapeau visible par chaque poste client (l'ordinateur de chaque utilisateur) ou chaque process sur un même poste. Un sémaphore a simplement pour rôle d'exister ou de ne pas exister. Chaque méthode exécutée par un utilisateur peut tester la présence d'un sémaphore. En créant et en testant des sémaphores, vous permettez aux méthodes de communiquer entre les postes clients et les process.

La fonction Semaphore retourne Vrai si sémaphore existe. Si sémaphore n'existe pas, Semaphore le crée et retourne Faux. Un seul utilisateur à la fois peut créer un sémaphore. Si Semaphore retourne Faux, cela indique que sémaphore n'existait pas, mais cela signifie également que sémaphore a été créé et positionné dans le process d'où l'appel a été effectué.

Semaphore retourne Faux si le sémaphore n'existait pas. La fonction retourne également Faux si le sémaphore avait été déjà positionné par le process d'où l'appel a été effectué.

Un sémaphore est limité à 255 caractères, métacaractères ($, <>) inclus. Si vous passez une chaîne plus longue, elle est tronquée. Attention, 4D tient compte de la casse des caractères en ce qui concerne les noms de sémaphores (le programme considère par exemple que "MonSémaphore" est différent de "monsémaphore").

Le paramètre optionnel nbTicks vous permet de spécifier un délai d'attente en ticks (1 tick = 1/60ème de seconde) si sémaphore est déjà positionné. Dans ce cas, avant de retourner Vrai, la fonction attend, dans la limite du temps fixé, que sémaphore se libère (auquel cas elle retourne Faux). Si le délai expire sans que sémaphore ait été libéré, Semaphore retourne Vrai.

Il y a deux types de sémaphores dans 4D : les sémaphores locaux et les sémaphores globaux. Un sémaphore local est visible par tous les process d'un même poste et seulement sur ce poste. Vous déclarez un sémaphore local en préfixant son nom avec le signe dollar ($). Les sémaphores locaux permettent de contrôler des opérations entre les différents process exécutés sur le même poste. Par exemple, un sémaphore local peut être utilisé pour gérer les accès à un tableau interprocess appelé par tous les process d'une base de données mono-utilisateur ou d'un poste client.
Un sémaphore global est visible par tous les utilisateurs et tous les process. Les sémaphores globaux permettent de contrôler des opérations entre les postes clients d'une base multi-utilisateurs.

Le principe de fonctionnement des sémaphores globaux et locaux est identique. Leur différence réside uniquement dans leur portée, c'est-à-dire leur visibilité. Dans 4D Server, les sémaphores globaux sont visibles pour tous les process de tous les postes clients. Un sémaphore local n'est visible que pour les process du poste client sur lequel il a été créé.

Avec 4D, les sémaphores globaux et locaux ont la même portée car il n'y a qu'un seul utilisateur. Cependant, si votre base est utilisée dans les deux environnements, n'hésitez pas à employer des sémaphores globaux et locaux, en fonction de vos besoins.

Les sémaphores ne servent pas à protéger l'accès aux enregistrements — cette gestion est effectuée automatiquement par 4D et 4D Server. Les sémaphores ont pour but d'éviter que plusieurs utilisateurs effectuent la même opération en même temps.

Exemples

(1) Dans l'exemple suivant, vous souhaitez empêcher que deux utilisateurs effectuent simultanément une mise à jour globale des prix dans une table [Produits]. Pour cela, des sémaphores sont utilisés :

```
Si (Semaphore("MAJPrix"))  ` Essai de création du sémaphore
   ALERTE("Un autre utilisateur est déjà en train de mettre à jour les prix. Essayez plus
                                                                                tard.")
Sinon
   MAJdesPrix  ` Méthode de mise à jour des prix
   EFFACER SEMAPHORE("MAJPrix"))  ` Effacer le sémaphore
Fin de si
```

(2) L'exemple suivant illustre l'utilisation d'un sémaphore local. Dans une base comportant plusieurs process, vous souhaitez maintenir une liste de "Choses à faire". Vous envisagez de la maintenir à jour dans un tableau interprocess et non dans une table. Vous devez empêcher les accès simultanés à l'aide d'un sémaphore. Dans ce cas, il vous suffit d'utiliser un sémaphore local car la liste "Choses à faire" est pour votre utilisation personnelle.

Le tableau interprocess est initialisé dans la méthode base Sur ouverture :

```
TABLEAU TEXTE (<>ListeAFaire;0) ` La liste de choses à faire est vide
```

Voici la méthode utilisée pour ajouter des éléments à la "liste des choses à faire" :

```
` Méthode projet AJOUTER LISTE A FAIRE
` AJOUTER LISTE A FAIRE (Texte)
` AJOUTER LISTE A FAIRE (Elément la liste à faire)

C_TEXTE($1)  ` Paramètre passé à la commande
Si (Non (Semaphore("$AccèsListe";300)))
        ` Attendre 5 secondes si un sémaphore existe déjà
    $vlElem:=Taille tableau(<>ListeAFaire)+1
    INSERER DANS TABLEAU(<>ListeAFaire;$vlElem)
    <>ListeAFaire{$vlElem}:=1
    EFFACER SEMAPHORE("$AccèsListe")  ` Effacer le sémaphore
Fin de si
```

Vous pouvez appeler cette méthode depuis n'importe quel process.

Référence

EFFACER SEMAPHORE, Tester semaphore.

Tester semaphore

version 6.5

Tester semaphore (sémaphore) → Booléen

Paramètre	Type		Description
sémaphore	Alpha	→	Nom du sémaphore à tester
Résultat	Booléen	←	Vrai = le sémaphore existe, Faux = le sémaphore n'existe pas

Description

La commande Tester semaphore permet de tester l'existence d'un sémaphore.

A la différence de la fonction Semaphore, Tester semaphore ne crée pas le sémaphore s'il n'existe pas.
Si le sémaphore existe, la fonction retourne Vrai, s'il n'existe pas elle retourne Faux.

Note : Attention, 4D tient compte de la casse des caractères en ce qui concerne les noms de sémaphores (le programme considère par exemple que "MonSémaphore" est différent de "monsémaphore").

Exemple

Cet exemple permet de connaître l'état d'un traitement (en l'occurrence, la modification d'un code) sans modifier le sémaphore :

```
Creer fenetre (x1;x2;y1;y2;-Fenêtre palette)
Repeter
   Si (Tester semaphore("Code d'encryptage"))
      POSITION MESSAGE ($x3;$y3)
      MESSAGE("Code d'encryptage en cours de modification.")
   Sinon
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Modification du code d'encryptage autorisée.")
   Fin de si
Jusque (StopInfo)
FERMER FENETRE
```

Référence

Semaphore.

VARIABLE VERS VARIABLE (process; varDestination; varSource{; varDestination2; varSource2; ...; varDestinationN; varSourceN})

Paramètre	Type		Description
process	Numérique	→	Numéro du process de destination
varDestination	Variable	→	Variable de destination
varSource	Variable	→	Variable source

Description

La commande VARIABLE VERS VARIABLE écrit la valeur de la ou des variable(s) varSource1 (varSource2, etc.), dans la ou les variable(s) process varDestination (varDestination2, etc.) du process de destination dont vous avez passé le numéro dans process.

VARIABLE VERS VARIABLE a un fonctionnement semblable à celui de la commande ECRIRE VARIABLE PROCESS, avec cependant les différences suivantes :
• Alors que vous passez comme source à ECRIRE VARIABLE PROCESS des expressions (et donc vous ne pouvez pas passer un tableau en totalité), vous devez passer comme source à VARIABLE VERS VARIABLE uniquement des variables (et donc vous pouvez passer un tableau en totalité).
• Avec ECRIRE VARIABLE PROCESS, chaque variable de destination peut être une variable ou un élément de tableau, mais ne peut pas être un tableau. Avec VARIABLE VERS VARIABLE, chaque variable de destination peut être une variable, un tableau ou un élément de tableau.

4D Server : La communication process “intermachine” permise par les commandes VARIABLE VERS VARIABLE, ECRIRE VARIABLE PROCESS et LIRE VARIABLE PROCESS n’est possible que du client vers le serveur. C’est toujours un process client qui lit ou écrit les variables d’une procédure stockée.

Pour chaque association varDestination;varSource, le type de la variable source doit être compatible avec la variable de destination, sinon vous pourrez obtenir des variables avec des valeurs non significatives. En mode interprété, si la variable de destination n'existe pas, elle est créée puis le type et la valeur de la variable source lui sont affectés.

Lorsque le process courant écrit les variables du process de destination, ce dernier n'est averti en aucune manière de l'écriture de l'instance de ses variables par un autre process.

Restrictions

VARIABLE VERS VARIABLE n'accepte pas de variables locales comme variables de destination.

VARIABLE VERS VARIABLE accepte tout type de variable process ou interprocess de destination, à l'exception de variables de type :
• Pointeur
• Tableau de pointeurs
• Tableau à deux dimensions

Le process de destination doit être un process utilisateur, ce ne peut être un des process du moteur de 4D. Si le process de destination n'existe pas, une erreur est retournée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.**Exemple**

L'exemple suivant récupère un tableau process depuis le process désigné par $vlProcess, passe séquentiellement tous ses éléments en caractères majuscules puis réécrit entièrement le tableau :

```
LIRE VARIABLE PROCESS($vlProcess;at_IPCom_Tab;$anTab)
Boucle($vlElem;1;Taille tableau($anTab))
    $anTab{$vlElem}:=Majusc($anTab{$vlElem})
Fin de boucle
VARIABLE VERS VARIABLE($vlProcess;at_IPCom_Tab;$anTab)
```

Référence

ECRIRE VARIABLE PROCESS, Introduction aux process, LIRE VARIABLE PROCESS.

45

Process (Interface utilisateur)

CACHER PROCESS (process)

Paramètre	Type		Description
process	Numérique	→	Numéro du process à cacher

Description

CACHER PROCESS masque toutes les fenêtres appartenant au process dont le numéro est process. Tous les éléments d'interface de process sont cachés jusqu'au MONTRER PROCESS suivant. La barre de menus du process est aussi cachée. L'ouverture d'une fenêtre alors que le process est caché ne provoquera aucun redessinement d'écran. Si le process est déjà caché, cette commande ne fait rien.

La seule exception à cette règle est la fenêtre du débogueur. Si la fenêtre du débogueur est affichée lorsque process est caché, process est affiché et passe au premier plan.

Si vous ne voulez pas qu'un process soit affiché lorsqu'il est créé, CACHER PROCESS doit être la première commande à appeler dans la méthode du process. Les process Process principal et Gestionnaire du cache ne peuvent pas être cachés à l'aide de cette commande.

Lorsqu'un process est caché, il est toujours en cours d'exécution.

Si vous souhaitez ne cacher qu'une fenêtre du process, utilisez la commande CACHER FENETRE.

Exemple

L'exemple suivant cachera toutes les fenêtres appartenant au process courant :

```
CACHER PROCESS (Numero du process courant)
```

Référence

MONTRER PROCESS, Statut du process.

MONTRER PROCESS (process)

Paramètre	Type		Description
process	Numérique	→	Numéro du process dont les fenêtres doivent être affichées

Description

MONTRER PROCESS fait apparaître l'ensemble des fenêtres appartenant à process. Cette commande ne passe pas les fenêtres de process au premier plan, utilisez pour cela la commande PASSER AU PREMIER PLAN.

Si les fenêtres de process sont déjà affichées, cette commande ne fait rien.

Exemple

L'exemple suivant affiche le process "Clients", s'il était caché auparavant. Le numéro de process est stocké dans la variable interprocess <>Clients :

```
MONTRER PROCESS (<>Clients)
```

Référence

CACHER PROCESS, PASSER AU PREMIER PLAN, Statut du process.

PASSER AU PREMIER PLAN (process)

Paramètre	Type		Description
process	Numérique	→	Numéro du process à passer au premier plan

Description

PASSER AU PREMIER PLAN passe les fenêtres du process de numéro process au premier plan. Toutes les fenêtres appartenant à process passent au premier plan. L'ordre des fenêtres est conservé. Si le process est déjà au premier plan, la commande ne fait rien. Si le process est caché, il faut utiliser la commande MONTRER PROCESS pour faire d'abord apparaître le process, sinon PASSER AU PREMIER PLAN ne fait rien.

Le Process principal et le Process de structure peuvent être passés au premier plan à l'aide de cette commande.

Exemple

L'exemple suivant est une méthode qui peut être exécutée à partir d'une commande de menu. Elle vérifie si le process au premier plan est le process <>Clients. Sinon, ce process passe au premier plan :

```
Si (Process de premier plan#<>Clients)  ` Si la liste des clients n'est pas affichée
   PASSER AU PREMIER PLAN (<>Clients)  ` Passer cette liste au premier plan
Fin de si
```

Référence

CACHER PROCESS, MONTRER PROCESS, Statut du process.

Process de premier plan {(*)} → Entier

Paramètre	Type		Description
*		→	Numéro du process de la première fenêtre non-flottante
Résultat	Entier	←	Numéro du process dont la ou les fenêtre(s) est (sont) au premier plan

Description

Process de premier plan retourne le numéro du process dont la ou les fenêtre(s) est (sont) au premier plan.

Lorsqu'une ou plusieurs fenêtres flottantes sont ouvertes, deux niveaux différents de fenêtres sont distingués :
• les fenêtres standard
• les fenêtres flottantes

Si la fonction Process de premier plan est utilisée dans la méthode formulaire ou dans une méthode objet d'une fenêtre flottante, la fonction retourne le numéro du process de la fenêtre flottante au premier plan parmi les fenêtres flottantes. Si vous passez le paramètre optionnel astérisque, la fonction retourne le numéro du process dont la fenêtre est au premier plan, **exception faite du niveau des fenêtres flottantes**.

Exemple

Référez-vous à l'exemple de PASSER AU PREMIER PLAN.

Référence

LISTE FENETRES, PASSER AU PREMIER PLAN.

46

Propriétés des objets

Propriétés des objets

version 6.5 (Modifiée)

Les commandes de propriétés des objets agissent sur les propriétés des objets présents dans les formulaires. Elles vous permettent de modifier l'apparence et le comportement de ces objets lorsque vous utilisez les formulaires pour afficher les enregistrements et en mode Application.

Important : La portée (l'aire d'action) de ces commandes est le formulaire en cours d'utilisation ; les modifications ne sont pas conservées lorsque vous sortez du formulaire.

Accès aux objets par leur nom d'objet ou à partir du nom de la source de données

Les commandes de propriétés des objets partagent toutes la même syntaxe :

NOM DE LA COMMANDE({*;} objet { ; paramètres spécifiques à la commande)

Si vous spécifiez le paramètre optionnel *, vous indiquez un nom d'objet (une chaîne) dans objet.

Vous pouvez utiliser le caractère @ dans ce nom si vous voulez adresser plusieurs objets du formulaire dans un seul appel. Le tableau qui suit montre des exemples de noms d'objets que vous pouvez spécifier pour cette commande.

Noms d'objets	Objets affectés par l'appel
zoneGroupe	Uniquement l'objet zoneGroupe.
zone@	Les objets dont le nom commence par “zone”.
@zoneGroupe	Les objets dont le nom finit par “zoneGroupe”.
@Groupe@	Les objets dont le nom contient “Groupe”.
zone@Btn	Les objets dont le nom commence par “zone” et finit par “Btn”.
@	Tous les objets présents dans le formulaire.

Note : Il est possible de paramétrer le mode d’interprétation du caractère @, lorsque celui-ci est inclus dans une chaîne de caractères. Cette option influe sur le fonctionnement des commandes du thème “Propriétés des objets”. Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Si vous omettez le paramètre optionnel *, vous indiquez un champ ou une variable dans objet. Dans ce cas, vous ne spécifiez pas une chaîne mais une référence de champ ou de variable (champ et variable objet uniquement).

Note : Si vous placez dans un formulaire plusieurs objets ayant des noms d'objets différents mais associés à une même variable, 4D appliquera toute commande à l'ensemble des objets, même si vous avez utilisé le nom d'objet comme paramètre. Par exemple, si un formulaire contient trois boutons nommés "Btn1", "Btn2" et "Btn3" associés à la même variable *var1*, l'exécution de l'instruction ACTIVER BOUTON(*;"Btn2") provoquera l'activation des trois boutons.

ACTIVER BOUTON ({*; }objet)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande ACTIVER BOUTON active le ou les objet(s) de formulaire désigné(s) par objet.

Un bouton ou un objet activé réagit aux clics souris et aux raccourcis clavier.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Cette commande (malgré ce que son nom suggère) peut être appliquée aux types d'objets suivants :

- Bouton, Bouton par défaut, Bouton 3D, Bouton invisible, Bouton inversé
- Bouton radio, Bouton radio 3D, Bouton image
- Case à cocher, Case à cocher 3D
- Pop-up menu, Liste déroulante, Combo Box, Menu/Liste déroulante
- Thermomètre, Règle

Note : Cette commande est sans effet avec un objet auquel une action standard a été assignée (4D se charge de modifier l'état de cet objet lorsque c'est nécessaire), sauf dans le cas des actions Valider et Annuler.

Exemples

(1) L'exemple suivant active le bouton bValider :

```
ACTIVER BOUTON(bValider)
```

(2) L'exemple suivant active tous les objets du formulaire dont le nom contient “btn”:

```
ACTIVER BOUTON(*;"@btn@")
```

(3) Reportez-vous l'exemple de la commande TITRE BOUTON.

Référence

INACTIVER BOUTON, TITRE BOUTON.

CHANGER JEU DE CARACTERES ({*; }objet; police)

Paramètre	Type		Description
*		→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet	→	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
police	Chaîne | Num	→	Nom ou numéro de police de caractères

Description

CHANGER JEU DE CARACTERES passe la police dans laquelle objet est affiché en police. Le paramètre police peut être soit un nom soit un numéro de police.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section d'introduction, Propriétés des objets.

Exemples

(1) L'exemple suivant définit la police d'un bouton nommé bOK. La police est Arial, une police système sous Windows :

```
CHANGER JEU DE CARACTERES (bOK; "Arial")  ` Modification de la police de MonBouton
```

(2) L'exemple suivant définit la police de tous les objets d'un formulaire dont le nom contient "info".

```
CHANGER JEU DE CARACTERES (*;"@info@";"Times")
```

(3) L'exemple suivant affecte la police de caractères spéciale %password, utilisable pour la saisie et l'affichage des champs de type “mots de passe” (les caractères sont masqués).

```
CHANGER JEU DE CARACTERES ([Utilisateurs]MotPasse;"%password")
```

Référence

CHANGER STYLE, CHANGER TAILLE.

CHANGER STYLE ({*; }objet; style)

Paramètre	Type		Description
*		→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet	→	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
style	Numérique	→	Style de police

Description

CHANGER STYLE assigne le style de police style à ou aux objet(s) de formulaire désigné(s) par objet.
Le nombre style est un code de style du système d'exploitation. En additionnant des codes, vous combinez les styles.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Vous devez passer dans le paramètre style une des constantes prédéfinies suivantes ou la somme de plusieurs de ces constantes :

Constante	Type	Valeur
Normal	Entier long	0
Gras	Entier long	1
Italique	Entier long	2
Souligné	Entier long	4

Exemples

(1) L'exemple suivant définit le style de police du bouton bAjoutNouveau. Le style demandé est gras italique :

```
CHANGER STYLE (bAjoutNouveau; Gras + Italique)
```

(2) L'exemple suivant définit le style de police Normal pour tous les objets de formulaire dont le nom débute par "vt" :

```
CHANGER STYLE (*; "vt@";Normal)
```

Référence

CHANGER JEU DE CARACTERES, CHANGER PROPRIETES ELEMENT, CHANGER TAILLE.

CHANGER TAILLE ({*; }objet; taille)

Paramètre	Type		Description
*		→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet	→	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
taille	Numérique	→	Taille de police en points

Description

CHANGER TAILLE définit la taille de la police du ou des objet(s) de formulaire spécifié(s) par objet.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La taille peut être tout Entier compris entre 1 et 255. Si la taille exacte n'existe pas, les caractères sont proportionnellement redimensionnés.

La zone de l'objet, telle qu'elle a été définie dans le formulaire, doit être suffisamment grande pour afficher les données dans la nouvelle taille. Autrement, le texte peut être tronqué ou pas du tout affiché.

Exemples

(1) L'exemple suivant définit la taille de police de la variable appelée vInfo :

```
CHANGER TAILLE (vInfo; 14)
```

(2) L'exemple suivant définit la taille de police de tous les objets de formulaire dont le nom débute par "hl" :

```
CHANGER TAILLE (*;"hl@";14)
```

Référence

CHANGER JEU DE CARACTERES, CHANGER STYLE.

CHOIX COULEUR ({*; }objet; couleur{; couleurAlt})

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Champ|Variable	→	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
couleur	Numérique	→	Nouvelles couleurs pour l'objet
couleurAlt	Numérique	→	Couleurs alternées pour une list box

Description

La commande CHOIX COULEUR définit les couleurs de premier plan et d'arrière-plan du ou des objet(s) de formulaire spécifié(s) par objet. Si objet est une list box, un paramètre supplémentaire permet de définir les couleurs de premier plan et d'arrière-plan des lignes paires (couleurs alternées).

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

couleurAlt permet de désigner une couleur alternative pour les lignes paires d'une list box ou d'une colonne de list box. Lorsque ce paramètre est passé, le paramètre couleur s'applique aux lignes impaires uniquement. Utiliser des couleurs alternées améliore la lisibilité des tableaux. Si objet désigne l'objet list box, les couleurs alternées sont utilisées dans la totalité de la list box. Si objet désigne une colonne, seule la colonne utilisera les couleurs définies.

Le paramètre couleur (ainsi que le paramètre couleurAlt) définit à la fois les couleurs de premier plan et d'arrière-plan. La couleur est calculée de la manière suivante : Couleur:=-(Premier_Plan+(256 * Arrière_Plan)), où Premier_Plan et Arrière_Plan sont des numéros de couleur (de 0 à 255) parmi la palette de couleurs de 4D — que vous pouvez visualiser, par exemple, dans la Liste des propriétés de l'éditeur de formulaires. Couleur est toujours un nombre négatif. Par exemple, si la couleur de premier plan est 20 et si la couleur d'arrière-plan est 10, alors couleur est égal à – (20 + (256 * 10)) soit –2580.

Les numéros les plus souvent utilisés sont fournis par 4D sous forme de constantes prédéfinies, placées dans le thème “Couleurs” :

Constante	Type	Valeur
Blanc	Entier long	0
Jaune	Entier long	1
Orange	Entier long	2
Rouge	Entier long	3
Violet	Entier long	4
Bleu foncé	Entier long	5
Bleu	Entier long	6
Bleu clair	Entier long	7
Vert	Entier long	8
Vert foncé	Entier long	9
Marron foncé	Entier long	10
Gris foncé	Entier long	11
Gris clair	Entier long	12
Marron	Entier long	13
Gris	Entier long	14
Noir	Entier long	15

Note : Tandis que CHOIX COULEUR travaille avec des couleurs indexées dans la palette de couleurs de 4D, la commande FIXER COULEURS RVB vous permet de travailler avec toute couleur RVB. Pour rétablir les couleurs automatiques d'un objet, utilisez la commande FIXER COULEURS RVB avec les constantes Coul premier plan et Coul arrière plan.

Exemple

L'exemple suivant définit la couleur de la zone de texte représentée ci-dessous dans l'éditeur de formulaires :

Après l'exécution de cette instruction :

```
CHOIX COULEUR (*;"Montexte"; - (Jaune + (256 * Rouge)))
```

... la zone prend l'apparence suivante :

Référence

FIXER COULEURS RVB.

CHOIX ENUMERATION

CHOIX ENUMERATION ({*; }objet; énum)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
énum	Alpha	→	Nom de l'énumération (définie en mode Structure)

Description

La commande CHOIX ENUMERATION définit ou remplace l'énumération associée à l'objet ou au groupe d'objets désigné(s) par objet avec l'énumération créée dans l'éditeur d'énumérations, en mode Développement, dont le nom est passé dans énum.

Cette commande peut être appliquée, dans un formulaire entrée ou un formulaire de dialogue, aux champs et variables saisissables dont les valeurs peuvent être saisies sous forme de texte. L'énumération s'affiche pendant la saisie lorsque l'utilisateur sélectionne la zone de texte.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Note : Cette commande ne peut pas être utilisée avec des champs placés dans le formulaire "liste" d'un sous-formulaire.

Exemple

L'exemple suivant définit l'énumération liée à un champ Coursiers. Si l'envoi doit être effectué de nuit, alors l'énumération affiche les sociétés de courses qui fonctionnent la nuit. Sinon, l'énumération standard est proposée :

```
Si ([Courses]Nuit)
    CHOIX ENUMERATION([Courses]Coursier; "Coursiers de nuit")
Sinon
    CHOIX ENUMERATION([Courses]Coursier; "Coursiers standard")
Fin de si
```

CHOIX FILTRE SAISIE

CHOIX FILTRE SAISIE ({*; }objet; filtreSaisie)

Paramètre	Type		Description
*		→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est un champ ou une variable
objet	Objet	→	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
filtreSaisie	Alpha	→	Nouveau filtre de saisie pour la zone saisissable

Description

CHOIX FILTRE SAISIE remplace le filtre de saisie pour objet par filtreSaisie dans le formulaire courant affiché à l'écran.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La commande CHOIX FILTRE SAISIE peut être utilisée dans des formulaires entrée et des dialogues et peut être appliquée aux champs et variables saisissables acceptant les filtres de saisie en mode Développement.

Pour enlever un filtre, passez une chaîne vide dans le paramètre filtreSaisie.

Note : Cette commande ne peut pas être utilisée avec des champs situés dans le formulaire "liste" d'un sous-formulaire.

Note : Pour pouvoir exploiter les filtres de saisie que vous avez créés dans la Boîte à outils, préfixez le nom du filtre, dans le paramètre filtreSaisie, d'une barre verticale (|).

Exemples

(1) L'exemple suivant définit le filtre de saisie pour le champ code postal. Si l'adresse se trouve en France, le filtre est paramétré pour les codes postaux français. Sinon, le filtre peut accepter toute valeur saisie :

```
Si (Pays = "France")  ` Fixer le filtre au format du code postal français
    CHOIX FILTRE SAISIE ([Sociétés]Code postal; &"#####")
Sinon  ` Fixer le filtre pour qu'il accepte toute valeur alphanumérique
    CHOIX FILTRE SAISIE ([Sociétés]Code postal; "~@")
Fin de si
```

(2) L'exemple suivant autorise uniquement la saisie des lettres “a”, “b”, “c” ou “g” dans un champ comportant deux lettres :

```
CHOIX FILTRE SAISIE([Table]Champ;"&"+Caractere (Guillemets)+"a;b;c;g"+
                                                Caractere (Guillemets)+"##")
```

Note : Cet exemple définit le filtre de saisie &"a;b;c;g"##.

Référence

CHOIX FORMATAGE.

CHOIX FORMATAGE ({*; }objet; formatAffich)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
formatAffich	Alpha	→	Nouveau format d'affichage de l'objet

Description

CHOIX FORMATAGE remplace le format d'affichage du ou des objet(s) spécifié(s) par objet avec le format que vous avez passé dans formatAffich. Le nouveau format est utilisé uniquement pour l'affichage courant, il n'est pas stocké avec le formulaire.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

La commande CHOIX FORMATAGE peut être indifféremment utilisée dans des formulaires entrée ou sortie (affichés ou imprimés) et appliquée aux champs ou aux variables (saisissables ou non saisissables). Bien entendu, vous devez utiliser un format d'affichage compatible avec le type de données présentes dans l'objet ou avec l'objet lui-même.

Booléens

Pour formater des champs booléens, vous disposez de deux possibilités :

• vous pouvez passer une valeur simple dans formatAffich. Dans ce cas, le champ sera affiché sous forme de case à cocher, son libellé sera la valeur définie.

• vous pouvez passer deux valeurs séparées par un point-virgule (;) dans formatAffich. Dans ce cas, le champ sera affiché sous forme de deux boutons radio.

Dates
Pour formater des champs ou variables de type Date, passez Caractere(n) dans formatAffich, où n peut être une des constantes prédéfinies suivantes (thème Formats d'affichage des dates) :

Constante	Type	Valeur
Système date court	Entier long	1
Système date abrégé	Entier long	2
Système date long	Entier long	3
Interne date court spécial	Entier long	4
Interne date long	Entier long	5
Interne date abrégé	Entier long	6
Interne date court	Entier long	7
ISO Date	Entier long	8
Vide si date nulle	Entier long	100

Note : La constante Vide si date nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit afficher une zone vide au lieu de zéros.

Heures
Pour formater des champs ou variables de type Heure, passez Caractere(n) dans formatAffich, où n peut être une des constantes prédéfinies suivantes (thème Formats d'affichage des heures) :

Constante	Type	Valeur
h mn s	Entier long	1
h mn	Entier long	2
Heures Minutes Secondes	Entier long	3
Heures Minutes	Entier long	4
h mn Matin Après Midi	Entier long	5
mn s	Entier long	6
Minutes secondes	Entier long	7
ISO Heure	Entier long	8
Système heure court	Entier long	9
Système heure long abrégé	Entier long	10
Système heure long	Entier long	11
Vide si heure nulle	Entier long	100

Note : La constante Vide si heure nulle doit être additionnée au format, elle indique qu'en cas de valeur nulle 4D doit afficher une zone vide au lieu de zéros.

Images
Pour formater des champs ou variables de type Image, passez Caractere(n) dans formatAffich, où n peut être une des constantes prédéfinies suivantes (thème Formats d'affichage des images) :

Constante	Type	Valeur
Tronquée centrée	Entier long	1
Non tronquée	Entier long	2
Sur fond	Entier long	3
Tronquée non centrée	Entier long	4
Proportionnelle	Entier long	5
Proportionnelle centrée	Entier long	6
Mosaïque	Entier long	7

Alphas et numériques
Pour formater des champs ou variables de type alpha ou numérique, passez directement le libellé du format dans le paramètre formatAffich.

Pour plus d'informations sur les formats d'affichage, reportez-vous au manuel *Mode Développement* de 4D.

Note : Pour pouvoir exploiter les formats d'affichage personnalisés que vous avez créés dans la boîte à outils, préfixez le nom du format, dans le paramètre formatAffich, d'une barre verticale (|).

Boutons image
Pour formater des boutons image, passez dans le paramètre formatAffich une chaîne de caractères respectant la syntaxe suivante :
cols;lignes;image;mode{;ticks}

• cols = nombre de colonnes de l'image
• lignes = nombre de lignes de l'image
• image = image utilisée, provenant de la bibliothèque d'images, d'une variable image ou d'une ressource PICT :
- si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250").
- si l'image provient d'une variable image, saisissez le nom de la variable
- si l'image provient d'une ressource PICT, saisissez son numéro, précédé de deux points (ex. : ":62500")
• mode = mode d'affichage et de fonctionnement du bouton image. Ce paramètre peut prendre les valeurs 0, 1, 2, 16, 32, 64 et 128, chaque valeur représentant un mode d'affichage ou de fonctionnement. Ces valeurs sont cumulatives ; en d'autres termes, pour sélectionner les valeurs 64 et 1, passez 65 dans le paramètre mode. Voici le détail de chaque valeur :

- mode = 0 (pas d'option)
Affiche l'image suivante de la série lorsque l'utilisateur clique sur le bouton. Affiche l'image précédente de la série lorsque l'utilisateur effectue Maj+clic sur le bouton. La séquence d'images s'arrête lorsqu'on atteint la dernière image de la série. En d'autres termes, le bouton ne retourne pas à la première image de la série.
- mode = 1 (Défilement continu sur clic)
Similaire au précédent, à la différence près que lorsque l'utilisateur clique sur l'image et maintient le bouton de la souris enfoncé, l'enchaînement des images est continu (c'est-à-dire que la série défile comme une animation). La séquence d'images s'arrête lorsqu'on atteint la dernière image de la série.
- mode = 2 (Recommencer la séquence)
Similaire au précédent, à la différence près que le défilement des images est "rebouclé" lorsqu'on atteint la dernière image de la séquence de défilement : une fois la dernière image atteinte, la première image est de nouveau affichée et la séquence recommence.
- mode = 16 (Bascule sur passage du curseur)
Le contenu du bouton image est modifié lorsque le curseur de la souris passe au-dessus de lui, sans que l'utilisateur ne clique. L'image initiale est rétablie lorsque le curseur quitte la zone du bouton. Ce mode, aussi appelé "Roll over", est fréquemment utilisé dans les navigateurs Web et dans les applications multimedia. L'image affichée est la dernière du tableau d'imagettes, sauf si le mode 128 (Dernière imagette si désactivé) est également sélectionné — dans ce cas, c'est l'avant-dernière imagette qui est utilisée comme "bascule".
- mode = 32 (Retour sur relâchement du clic)
Ce mode fonctionne avec deux images ; il indique que le bouton doit toujours afficher la première image, sauf quand l'utilisateur clique dessus. En d'autres termes, le bouton affiche l'image A par défaut, l'image B lorsqu'il reçoit un clic souris, et de nouveau l'image A dès que le bouton de la souris est relâché. Ce mode permet de réaliser un bouton d'action avec une image différente pour chaque état (normal et enfoncé). Vous pouvez ainsi créer un effet 3D personnalisé ou toute image symbolisant l'action effectuée par bouton.
- mode = 64 (Transparent)
Permet de rendre transparent le fond de l'image.
- mode = 128 (Dernière imagette si désactivé)
Permet d'indiquer que la dernière image de la série doit être utilisée lorsque le bouton est inactivé. Avec ce paramétrage, 4D affiche la dernière "partie" de l'image référencée lorsque le bouton image est inactivé. L'image d'inactivation est traitée à part par 4D : lorsque vous combinez cette option avec les valeurs 0, 1 ou 2 dans le paramètre mode, la dernière image est exclue de la séquence associée au bouton et n'apparaîtra que lorsqu'il sera inactivé.
• ticks = activation du mode "défilement automatique tous les N ticks" et intervalle de temps séparant l'affichage de chaque image. Ce paramètre optionnel, s'il est passé, provoque le défilement automatique et en boucle du contenu du bouton image à la vitesse spécifiée. Par exemple, si vous passez "2;3;?16807;0;10", la variation du bouton image s'effectuera tous les 10 ticks. Dans ce mode, toutes les autres options sont ignorées — à l'exception de l'option "Transparent" (mode 64).

Pop up menus image
Pour formater des pop up menus image, passez dans le paramètre formatAffich une chaîne de caractères respectant la syntaxe suivante :
cols;lignes;image;margeH;margeV;mode

• cols = nombre de colonnes de l'image
• lignes = nombre de lignes de l'image
• image = image utilisée, provenant de la bibliothèque d'images, d'une variable image ou d'une ressource PICT :
- si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250").
- si l'image provient d'une variable image, saisissez le nom de la variable
- si l'image provient d'une ressource PICT, saisissez son numéro, précédé de deux points (ex. : ":62500")
• margeH = marge en pixels entre les limites horizontales du menu et l'image.
• margeV = marge en pixels entre les limites verticales du menu et l'image.
• mode = mode de transparence du pop up menu image. Accepte les valeurs 0 et 64 :
- mode = 0 : le pop-up menu image n'est pas transparent,
- mode = 64 : le pop-up menu image est transparent.

Thermomètres et règles
Pour formater des objets de type thermomètre ou règle, passez dans le paramètre formatAffich une chaîne de caractères respectant la syntaxe suivante :
min;max;unité;pas;mode{;format}

• min = valeur de la graduation d'origine de la jauge
• max = valeur de la graduation de fin de la jauge
• unité = intervalle entre les graduations de la jauge
• pas = intervalle de déplacement du curseur dans la jauge
• mode = mode d'affichage et de fonctionnement de la jauge. Ce paramètre accepte les valeurs 0, 2, 3, 16, 32 et 128. Ces valeurs peuvent être cumulées afin de définir plusieurs options (hormis le mode 128). Voici le détail de chaque valeur :
- mode = 0 : ne pas afficher les libellés
- mode = 2 : afficher les libellés à droite ou au-dessous de la jauge
- mode = 3 : afficher les libellés à gauche ou au-dessus de la jauge
- mode = 16 : afficher les graduations en regard des libellés
- mode = 32 : déclencher la méthode objet avec l'événement Sur données modifiées pendant que l'utilisateur change la valeur de la jauge. Par défaut, la méthode est exécutée après la modification.
- mode = 128: activer le mode "Barber shop" (animation continue). Cette valeur ne peut pas être cumulée. Dans ce mode, les autres paramètres sont ignorés. Pour plus d'informations sur ce mode, reportez-vous au manuel *Développement*.

• format = format d'affichage des graduations de la jauge.
A noter les libellés et les graduations sont automatiquement masqués si la taille de l'objet jauge ne permet pas de les afficher correctement.

Cadrans
Pour formater des objets de type cadran, passez dans le paramètre formatAffich une chaîne de caractères respectant la syntaxe suivante :
min;max;unité;pas{;mode}

• min = valeur de la graduation d'origine du cadran
• max = valeur de la graduation de fin du cadran
• unité = intervalle entre les graduations du cadran
• pas = intervalle de déplacement du curseur dans le cadran
• mode = mode de fonctionnement du cadran (facultatif). Ce paramètre accepte uniquement la valeur 32 : déclencher la méthode objet avec l'événement Sur données modifiées pendant que l'utilisateur change la valeur du cadran. Par défaut, la méthode est exécutée après la modification.

Grilles de boutons
Pour formater des grilles de boutons, passez dans le paramètre formatAffich une chaîne de caractères respectant la syntaxe suivante :
cols;lignes

• cols = nombre de colonnes de la grille
• lignes = nombre de lignes de la grille

Note : Pour plus d'informations sur les formats d'affichage des objets de formulaire, reportez-vous au manuel *Mode Développement* de 4D.

Boutons 3D
Pour formater des boutons 3D, passez dans le paramètre formatAffich une chaîne de caractères respectant la syntaxe suivante :
titre;image;imageFond;posTitre;titreVisible;icôneVisible;style;margeHor;margeVert;décalageIcône;popupMenu

• titre = titre du bouton. Cette valeur peut être exprimée sous forme de texte ou de numéro de ressource (ex. : ":16800,1")
• image = image associée au bouton, provenant de la bibliothèque d'images, d'une variable image ou d'une ressource PICT :
- si l'image provient de la bibliothèque d'images, saisissez son numéro, précédé d'un point d'interrogation (ex. : "?250").
- si l'image provient d'une variable image, saisissez le nom de la variable.

- si l'image provient d'une ressource PICT, saisissez son numéro, précédé de deux points (ex. : ":62500").
- si l'image provient d'un fichier stocké dans le dossier Resources de la base, saisissez un URL du type "#{dossier/}nomimage" ou "file:{dossier/}nomimage".

• imageFond = image de fond associée au bouton (style Personnalisé), provenant de la bibliothèque d'images, d'une variable image, d'une ressource PICT ou d'un fichier stocké dans le dossier Resources (cf. ci-dessus).

• posTitre = position du titre du bouton. Cinq valeurs sont possibles :
- posTitre = 1 : Gauche
- posTitre = 2 : Haut
- posTitre = 3 : Droite
- posTitre = 4 : Bas
- posTitre = 5 : Centre

• titreVisible = Titre visible ou non. Deux valeurs sont possibles :
- titreVisible = 0 : le titre est masqué
- titreVisible = 1 : le titre est affiché

• icôneVisible = Icône visible ou non. Deux valeurs sont possibles :
- icôneVisible = 0 : l'icône est masquée
- icôneVisible = 1 : l'icône est affichée

• style = Style du bouton. La valeur de cette option détermine la prise en compte de certaines autres options (par exemple imageFond). Dix valeurs de style sont possibles :
- style = 0 : Aucun
- style = 1 : Décalage du fond
- style = 2 : Bouton poussoir
- style = 3 : Bouton barre outils
- style = 4 : Personnalisé
- style = 5 : Rond
- style = 6 : Petit carré système
- style = 7 : Office XP
- style = 8 : Bevel
- style = 9 : Bevel arrondi

• margeHor = Marge horizontale. Nombre de pixels délimitant les marges internes à droite et à gauche du bouton (zones dans lesquelles l'icône et le texte ne doivent pas empiéter).

• margeVert = Marge verticale. Nombre de pixels délimitant les marges internes en haut et en bas du bouton (zones dans lesquelles l'icône et le texte ne doivent pas empiéter).

• décalageIcône = Décalage de l'icône vers la droite et le bas. Cette valeur, exprimée en pixels, indique le décalage de l'icône du bouton vers la droite et le bas en cas de clic (la même valeur est utilisée pour les deux directions).

• popupMenu = Association d'un pop up menu au bouton. Trois valeurs sont possibles :
- popupMenu = 0 : Sans pop up menu
- popupMenu = 1 : Avec pop up menu lié
- popupMenu = 2 : Avec pop up menu séparé

Certaines options ne sont pas prises en charge dans tous les styles de boutons 3D. De plus, dans certains cas vous pourrez souhaiter ne pas modifier toutes les options. Pour ne pas passer une option, il suffit d'omettre la valeur correspondante. Par exemple, pour ne pas passer les options titreVisible et margeVert, vous pouvez écrire :

```
CHOIX FORMATAGE(maVar;"JoliBouton;?256;:562;1;;1;4;5;;5;0")
```

Exemples

(1) La ligne de code suivante formate le champ [Employés]Date embauche au cinquième format de date.

```
CHOIX FORMATAGE ([Employés]Date embauche; Caractere(Interne date long))
```

(2) L'exemple suivant change le format d'un champ [Sociétés]Code postal selon la longueur du code postal :

```
Si (Longueur ([Sociétés]Code postal) = 9)
    CHOIX FORMATAGE ([Sociétés]Code postal; "#####-####")
Sinon
    CHOIX FORMATAGE ([Sociétés]Code postal; "#####")
Fin de si
```

(3) L'exemple suivant définit le format d'un champ booléen pour afficher soit "Marié" soit "Célibataire" au lieu des valeurs par défaut "Oui" et "Non" :

```
CHOIX FORMATAGE ([Employés]Situation; "Marié;Célibataire")
```

(4) L'exemple suivant définit le format d'un champ booléen pour afficher une case à cocher libellée "Classé" :

```
CHOIX FORMATAGE ([Dossier]Classement; "Classé")
```

(5) Vous disposez d'un tableau d'imagettes contenant 1 ligne et 4 colonnes, destiné à afficher un bouton image ("actif par défaut", "bouton cliqué", "survol du curseur" et "inactivé"). Vous souhaitez lui associer les options Bascule sur passage du curseur, Retour sur relâchement du clic et Dernière imagette si désactivé :

```
CHOIX FORMATAGE (*;"BoutonImage"; "4;1;?15000;176")
```

(6) Passage d'un thermomètre en mode "Barber shop" :

```
CHOIX FORMATAGE ($Monthermo;";;;;128")
$Monthermo:=1 `Déclencher l'animation
```

Référence

CHOIX FILTRE SAISIE, Lire formatage, LIRE FORMATAGE SYSTEME.

CHOIX SAISISSABLE ({*; }objet; zoneSaisie)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une table, un champ ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * spécifié) ou Table ou Champ ou Variable (si * omis)
zoneSaisie	Booléen	→	Vrai = saisissable ; Faux = non saisissable

Description

CHOIX SAISISSABLE rend saisissable ou non saisissable le ou les objet(s) de formulaire désigné(s) par objet.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est une table, un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de table, de champ ou de variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

L'utilisation de cette commande est équivalente à la sélection de l'option Saisissable pour un champ ou une variable dans la Liste des propriétés de l'éditeur de formulaires. CHOIX SAISISSABLE fonctionne avec un sous-formulaire uniquement si elle se trouve dans la méthode formulaire du sous-formulaire.

Lorsque zoneSaisie est saisissable (Vrai), l'utilisateur peut y placer le curseur pour saisir des données. Lorsque zoneSaisie est non saisissable (Faux), l'utilisateur ne peut pas placer le curseur dans la zone et ne peut donc pas saisir de valeurs.

La commande CHOIX SAISISSABLE permet également d'activer par programmation le mode "Saisie en liste" pour les sous-formulaires et les formulaires liste affichés par les commandes MODIFIER SELECTION et VISUALISER SELECTION :
• Pour les sous-formulaires, vous pouvez passer dans le paramètre objet soit le nom de la table du sous-formulaire, soit le nom de l'objet sous-formulaire lui-même, par exemple : CHOIX SAISISSABLE(*;"Sousform";Vrai)

• Pour les formulaires liste, vous devez passer le nom de la table du formulaire dans le paramètre objet, par exemple : CHOIX SAISISSABLE([MaTable];Vrai).

Rendre un objet non saisissable n'empêche pas sa modification par programmation.

Exemples

(1) L'exemple suivant définit un champ de type d'expédition suivant le poids d'un colis expédié. Si le colis pèse un kilo ou moins, l'expéditeur sera La Poste et le champ est rendu non saisissable. Sinon, le champ est rendu saisissable.

```
Si ([Expédition]Poids <= 1)
    [Expédition]Type := "La Poste"
    CHOIX SAISISSABLE ([Expédition]Type; Faux)
Sinon
    CHOIX SAISISSABLE ([Expédition]Type; Vrai)
Fin de si
```

(2) Voici la méthode objet d'une case à cocher placée dans l'en-tête d'une liste pour contrôler le mode Saisie en liste :

```
C_BOOLEEN(bSaisissable)
CHOIX SAISISSABLE([Table1];bSaisissable)
```

Référence

ACTIVER BOUTON, CHOIX VISIBLE, INACTIVER BOUTON.

CHOIX VISIBLE ({*; }objet; visible)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * est passé) ou Champ ou Variable (si * est omis)
visible	Booléen	→	Vrai = visible, Faux = invisible

Description

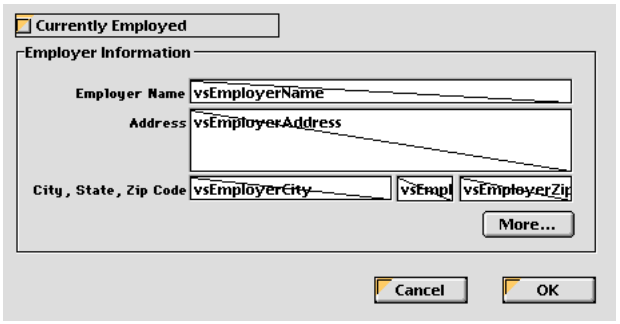
La commande CHOIX VISIBLE affiche ou masque le ou les objet(s) défini(s) par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre objet désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables objets uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Si vous passez la valeur VRAI dans le paramètre visible, le ou les objet(s) sont affichés. Si vous passez FAUX dans visible, les objets sont masqués.

Exemple

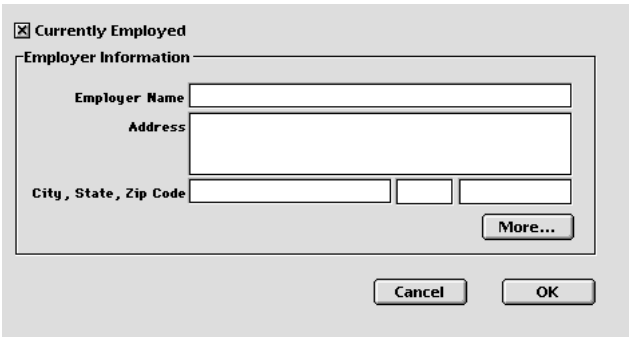
Voici un formulaire tel qu'il apparaît en mode Développement :



Les objets dans la zone de groupe **Employer Information** ont tous un nom qui contient l'expression “employer” (y compris la zone de groupe). Lorsque l'option **Currently Employed** est cochée, les objets doivent être visibles, lorsqu'elle est désélectionnée les objets doivent être invisibles. Voici la méthode projet de la case à cocher :

```
  ` Méthode objet Case à cocher cbCurrentlyEmployed
Au cas ou
    : (Evenement formulaire=Sur chargement)
        cbCurrentlyEmployed:=1

    : (Evenement formulaire=Sur clic)
            ` Cacher ou montrer tous les objets dont le nom contient "emp"
        CHOIX VISIBLE(*;"@emp@";cbCurrentlyEmployed # 0)
            ` Mais toujours conserver la case à cocher visible
        CHOIX VISIBLE(cbCurrentlyEmployed;Vrai)
Fin de cas
```

En exécution, le formulaire apparaîtra ainsi :



ou ainsi :

Référence

ACTIVER BOUTON, CHOIX SAISISSABLE, INACTIVER BOUTON.

CHOIX VISIBLE BARRES DEFILEMENT

CHOIX VISIBLE BARRES DEFILEMENT ({*; }objet; horizontal; vertical)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
horizontal	Booléen	→	Vrai=montrer, Faux=cacher
vertical	Booléen	→	Vrai=montrer, Faux=cacher

Description

La commande CHOIX VISIBLE BARRES DEFILEMENT permet d'afficher ou de masquer les barres de défilement horizontale et/ou verticale dans l'objet désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable. Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Cette commande est utilisable avec les objets de formulaire suivants :
- list box,
- zones de défilement,
- listes hiérarchiques,
- sous-formulaires.

Passez dans les paramètres horizontal et vertical des valeurs booléennes indiquant si les barres de défilement correspondantes doivent être affichées (Vrai) ou cachées (Faux). Par défaut, les barres de défilement sont affichées.

Note : Les objets de type zone de défilement ne sont pas pourvus de barres de défilement horizontales. Le paramètre horizontal étant toutefois obligatoire, vous devez le passer dans ce cas, il sera simplement ignoré.

Référence

CHOIX VISIBLE, Lire information listbox, MONTRER GRILLE LISTBOX.

DEPLACER OBJET ({*; }objet; dépH; dépV{; redimH{; redimV{; *}}})

Paramètre	Type		Description
*	*	→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est une variable
objet	Objet	→	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
dépH	Entier long	→	Valeur de déplacement horizontal de l'objet (>0 = vers la droite, <0 = vers la gauche)
dépV	Entier long	→	Valeur de déplacement vertical de l'objet (>0 = vers le bas, <0 = vers le haut)
redimH	Entier long	→	Valeur de redimensionnement horizontal de l'objet
redimV	Entier long	→	Valeur de redimensionnement vertical de l'objet
*	*	→	Si spécifié = coordonnées absolues Si omis = coordonnées relatives

Description

La commande DEPLACER OBJET permet de déplacer le ou les objet(s) du formulaire courant, défini(s) par les paramètres * et objet, de dépH pixels horizontalement et de dépV pixels verticalement.
Il est également possible (optionnellement) de redimensionner le ou les objet(s) de redimH pixels horizontalement et de redimV pixels verticalement.

Le sens du déplacement ou du redimensionnement est défini par le signe des valeurs passées dans les paramètres dépH et dépV :
• Si la valeur est positive, le déplacement ou le redimensionnement s'effectue respectivement vers la droite ou vers le bas.
• Si la valeur est négative, le déplacement ou le redimensionnement s'effectue respectivement vers la gauche ou vers le haut.

Si vous passez le premier paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable de type objet uniquement).
Si vous passez un nom d'objet dans le paramètre objet et utilisez le caractère joker @ afin de sélectionner plusieurs objets, tous les objets sélectionnés seront déplacés ou redimensionnés.

Par défaut, les valeurs de dépH, dépV, redimH et redimV modifient les coordonnées de l'objet relativement à sa position précédente. Si vous souhaitez que ces paramètres définissent des coordonnées absolues, passez le dernier paramètre optionnel *.

Cette commande fonctionne uniquement dans les contextes suivants :
• Formulaires entrée en mode saisie,
• Formulaires affichés via la commande DIALOGUE,
• En-têtes et pieds de page des formulaires sortie affichés par les commandes MODIFIER SELECTION ou VISUALISER SELECTION,
• Formulaires en cours d'impression.

Exemples

(1) L'instruction suivante déplace le bouton "Bouton_1" de 10 pixels vers la droite et de 20 pixels vers le haut, et agrandit le bouton de 30 pixels en largeur et de 40 en hauteur :

```
DEPLACER OBJET (*;"Bouton_1";10;-20;30;40)
```

(2) L'instruction suivante place le bouton "Bouton_1" aux coordonnées (10;20) (30;40) :

```
DEPLACER OBJET (*;"Bouton_1";10;20;30;40;*)
```

Référence

LIRE RECT OBJET.

FIXER ALIGNEMENT ({*; }objet; alignement)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de fomulaire	→	Nom d'objet (si * est passé) ou Champ ou variable (si * est omis)
alignement	Numérique	→	Code d'alignement

Description

La commande FIXER ALIGNEMENT vous permet de fixer le type d'alignement appliqué à l'objet ou aux objets désigné(s) par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre objet désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables de formulaire uniquement).

Passez dans le paramètre alignement une des constantes du thème "Alignement objet" :

Constante	Type	Valeur
Aligné par défaut	Entier long	1
Aligné à gauche	Entier long	2
Centré	Entier long	3
Aligné à droite	Entier long	4

Les objets de formulaire auxquels vous pouvez appliquer cette commande sont les suivants :

- Zones de défilement
- Combo box
- Textes statiques
- Zones de groupes
- Pop up menus/Listes déroulantes
- Champs
- Variables

Référence

Lire alignement.

FIXER COULEURS RVB ({*; }objet; couleurAvantPlan; couleurArrièrePlan{; couleurArrièrePlanAlt})

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet	→	Nom d'objet (si * est spécifié) ou Champ ou Variable (si * est omis)
couleurAvantPlan	Numérique	→	Valeur de la couleur RVB d'avant-plan
couleurArrièrePlan	Numérique	→	Valeur de la couleur RVB d'arrière-plan
couleurArrièrePlanAlt	Numérique	→	Valeur de la couleur RVB d'arrière-plan alternée

Description

La commande FIXER COULEURS RVB modifie les couleurs d'avant-plan et d'arrière-plan du ou des objet(s) défini(s) par le paramètre objet et le paramètre optionnel *. Lorsque la commande est appliquée à un objet de type List box, un paramètre supplémentaire permet de modifier la couleur alternée des lignes.

Si vous passez le paramètre optionnel *, vous spécifiez que le paramètre objet est le nom d'un objet (une chaîne de caractères). Si le paramètre * est omis, vous spécifiez que objet est un champ ou un objet. Dans ce cas, vous ne passez pas dans objet une chaîne de caractères mais la référence à un champ ou à une variable (champ ou variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Le paramètre facultatif couleurArrièrePlanAlt permet de désigner une couleur alternative pour l'arrière-plan (c'est-à-dire le fond) des lignes paires. Ce paramètre n'est utile que lorsque l'objet désigné est de type List box ou colonne de list box. Lorsque ce paramètre est utilisé, la couleurArrièrePlan est utilisée pour le fond des lignes impaires uniquement. Utiliser des couleurs alternées améliore la lisibilité des tableaux.
Si objet désigne l'objet List box, les couleurs alternées sont utilisées dans la totalité de la list box. Si objet désigne une colonne de list box, seule la colonne utilisera les couleurs définies.

Vous passez des valeurs de couleurs RVB dans les paramètres couleurAvantPlan, couleurArrièrePlan et couleurArrièrePlanAlt. Ces valeurs sont des entiers longs de 4 octets dont le format (0x00RRGGBB) est décrit ci-dessous (les octets sont numérotés de 0 à 3 de la droite vers la gauche) :

Octet	Description
3	Doit être zéro pour une couleur RVB absolue
2	Composante rouge de la couleur (0..255)
1	Composante verte de la couleur (0..255)
0	Composante bleue de la couleur (0..255)

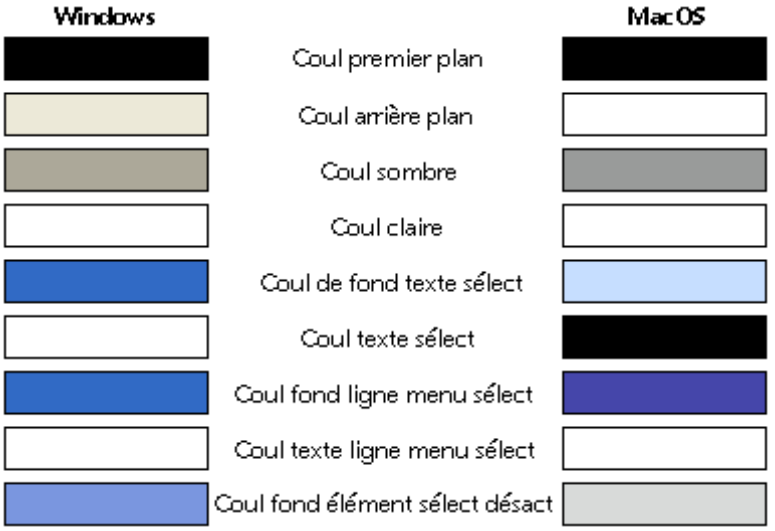
Le tableau ci-dessous présente des exemples de valeurs de couleurs RVB :

Valeur	Description
0x00000000	Noir
0x00FF0000	Rouge vif
0x0000FF00	Vert vif
0x000000FF	Bleu vif
0x007F7F7F	Gris
0x00FFFF00	Jaune vif
0x00FF7F7F	Rouge pastel
0x00FFFFFF	Blanc

Vous pouvez aussi spécifier une des couleurs “système” utilisées par défaut par 4D pour dessiner des objets ayant la propriété de couleur “automatique”. Les constantes prédéfinies suivantes sont proposées par 4D dans le thème “FIXER COULEURS RVB” :

Constante	Type	Valeur
Coul premier plan	Entier long	-1
Coul arrière plan	Entier long	-2
Coul sombre	Entier long	-3
Coul claire	Entier long	-4
Coul de fond texte sélect	Entier long	-7
Coul texte sélect	Entier long	-8
Coul fond ligne menu sélect	Entier long	-9
Coul texte ligne menu sélect	Entier long	-10
Coul fond élément sélect désact	Entier long	-11

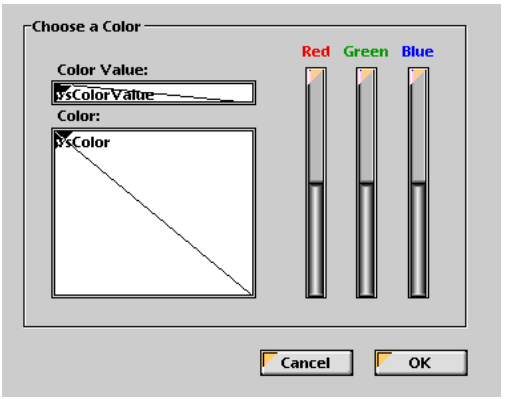
Ces couleurs (sur un système standard) sont les suivantes :



ATTENTION : Notez que, sous Windows, ces couleurs automatiques dépendent du système. Si vous modifiez vos couleurs système dans le Panneau de configuration "Affichage", les couleurs automatiques de 4D seront modifiées en conséquence. Utilisez les valeurs de couleurs automatiques pour assigner à des objets les couleurs système, et non pour leur assigner les mêmes couleurs que celles définies dans l'exemple ci-dessus.

Exemple

Voici un formulaire contenant deux variables non saisissables, vsColorValue et vsColor ainsi que trois thermomètres, thRouge, thVert et thBleu :



Les méthodes associées à ces objets sont les suivantes :

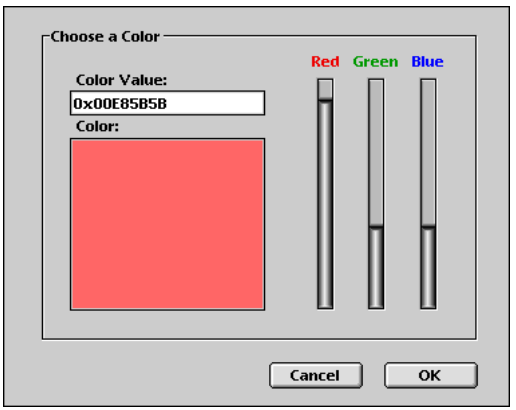
```
      ` Méthode objet de la variable non saisissable vsColorValue
Au cas ou
      : (Evenement formulaire=Sur chargement)
            vsColorValue:="0x00000000"
Fin de cas

      ` Méthode objet de la variable non saisissable vsColor
Au cas ou
      : (Evenement formulaire=Sur chargement)
            vsColor:=""
            FIXER COULEURS RVB(vsColor;0x00FFFFFF;0x0000)
Fin de cas

      ` Méthode objet du thermomètre thRouge
CLIC SUR THERMOMETRE COULEUR

      ` Méthode objet du thermomètre thVert
CLIC SUR THERMOMETRE COULEUR

      ` Méthode objet du thermomètre thBleu
CLIC SUR THERMOMETRE COULEUR
```

La méthode projet appelée par les trois thermomètres est la suivante :

```
` Méthode projet CLIC SUR THERMOMETRE COULEUR
FIXER COULEURS RVB(vsColor;0x00FFFFFF;(thRouge << 16)+(thVert << 8)+thBleu)
vsColorValue:=Chaine((thRouge << 16)+(thVert << 8)+thBleu;"&x")
Si (thRouge=0)
   vsColorValue:=Sous chaine(vsColorValue;1;2)+"0000"+Sous chaine(vsColorValue;3)
Fin de si
```

Notez l'utilisation des Opérateurs sur les bits pour le calcul des valeurs des couleurs à partir de celles des thermomètres.

En exécution, le formulaire a l'aspect suivant :



Référence

CHOIX COULEUR, Opérateurs sur les bits, Selectionner couleur RVB.

INACTIVER BOUTON ({*; }objet)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande INACTIVER BOUTON inactive le ou les objet(s) de formulaire désigné(s) par objet.

Un bouton ou un objet inactivé ne réagit pas aux clics souris ni aux raccourcis clavier, et est affiché en grisé.

Note : Désactiver un bouton ou un objet ne vous empêche pas de modifier sa valeur par programmation.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

Cette commande (malgré ce que son nom suggère) peut être appliquée aux types d'objets suivants :

- Bouton, Bouton par défaut, Bouton 3D, Bouton invisible, Bouton inversé
- Bouton radio, Bouton radio 3D, Bouton image
- Case à cocher, Case à cocher 3D
- Pop-up menu, Liste déroulante, Combo Box, Menu/Liste déroulante
- Thermomètre, Règle

Note : Cette commande est sans effet avec un objet auquel une action standard a été assignée (4D se charge de modifier l'état de cet objet lorsque c'est nécessaire), sauf dans le cas des actions Valider et Annuler.

Exemples

(1) L'exemple suivant inactive le bouton bValider :

```
INACTIVER BOUTON(bValider)
```

(2) L'exemple suivant inactive tous les objets de formulaire dont le nom contient “btn” :

```
INACTIVER BOUTON(*;"@btn@")
```

(3) Reportez-vous à l'exemple de la commande TITRE BOUTON.

Référence

ACTIVER BOUTON, TITRE BOUTON.

Lire alignement ({*; }objet) → Numérique

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * est passé) ou Champ ou variable (si * est omis)
Résultat	Numérique	←	Code d'alignement

Description

La commande Lire alignement retourne un code indiquant le type d'alignement appliqué à l'objet désigné par les paramètres objet et *.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet désigne le nom d'un objet (une chaîne). Si vous ne passez pas le paramètre *, vous indiquez que le paramètre objet désigne un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne de caractères mais la référence du champ ou de la variable (champs ou variables de formulaire uniquement).

Note : Si vous appliquez la commande à un ensemble d'objets, seule la valeur d'alignement du dernier objet est retournée.

Le code retourné correspond à l'une des constantes du thème "Alignement objet" :

Constante	Type	Valeur
Aligné par défaut	Entier long	1
Aligné à gauche	Entier long	2
Centré	Entier long	3
Aligné à droite	Entier long	4

Les objets de formulaire auxquels un alignement peut être appliqué sont les suivants :
- Zones de défilement
- Combo box
- Textes statiques
- Zones de groupes
- Pop up menus/Listes déroulantes
- Champs
- Variables

Référence

FIXER ALIGNEMENT.

Lire formatage ({*; }objet) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est un champ ou une variable
objet	Objet de formulaire	→	Nom d'objet (si * spécifié) ou Champ ou variable (si * omis)
Résultat	Chaîne	←	Format d'affichage de l'objet

Description

La commande Lire formatage retourne le format d'affichage courant appliqué à l'objet spécifié par le paramètre objet.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (dans ce cas, passez une chaîne dans objet). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable.

Cette commande retourne le format d'affichage courant de l'objet, c'est-à-dire le format défini en mode Développement ou à l'aide de la commande CHOIX FORMATAGE. Lire formatage fonctionne avec tous les types d'objets de formulaire (champs ou variables) acceptant un format d'affichage : booléen, date, heure, image, chaîne, numérique, ainsi que les grilles de boutons, cadrans, thermomètres, règles, pop up menus image, boutons image et boutons 3D. Pour plus d'informations sur les formats d'affichage de ces objets, reportez-vous à la documentation de la commande CHOIX FORMATAGE.

Note : Si vous appliquez la commande à un ensemble d'objets, seul le formatage du dernier objet pris en compte est retourné.

Lorsque la commande Lire formatage est appliquée à des objets de type date, heure ou image (formats définis sous forme de constantes), la chaîne retournée correspond au code de caractere de la constante. Pour obtenir la valeur de la constante, il suffit d'appliquer la fonction Code de caractere au résultat (cf. exemple ci-dessous).

Exemples

(1) Cet exemple permet d'obtenir la valeur de la constante de formatage appliquée à la variable image dont le nom d'objet est "maphoto" :

```
C_ALPHA(2;$format)
CHOIX FORMATAGE(*;"maphoto";Caractere(Sur fond))
    `Application du format sur fond (valeur = 3)
$format:=Lire formatage(*;"maphoto")
ALERTE("Format numéro :"+Chaine(Code de caractere($format)))
    `Affichage de la valeur "3"
```

(2) Cet exemple permet d'obtenir le formatage appliqué au champ booléen [Adhérents]Etat_civil :

```
C_ALPHA(30;$format)
$format:=Lire formatage([Adhérents]Etat_civil)
ALERTE($format)  `Affichage du format, par exemple "Marié;Célibataire"
```

Référence

CHOIX FORMATAGE.

LIRE RECT OBJET ({*; }objet; gauche; haut; droite; bas)

Paramètre	Type		Description
*	*	→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est une variable
objet	Objet	→	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis)
gauche	Entier long	←	Coordonnée gauche de l'objet
haut	Entier long	←	Coordonnée supérieure de l'objet
droite	Entier long	←	Coordonnée droite de l'objet
bas	Entier long	←	Coordonnée inférieure de l'objet

Description

La commande LIRE RECT OBJET retourne dans les variables ou champs gauche, haut, droite et bas les coordonnées (en points) du ou des objet(s) du formulaire courant défini(s) par les paramètres * et objet.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable de type objet uniquement).
Si vous passez un nom d'objet dans le paramètre objet et utilisez le caractère joker @ afin de sélectionner plusieurs objets, les coordonnées retournées seront celles du rectangle formé par l'ensemble des objets concernés.

Note : Il est possible de paramétrer le mode d'interprétation du caractère @, lorsque celui-ci est inclus dans une chaîne de caractères. Cette option influe sur le fonctionnement des commandes du thème "Propriétés des objets". Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Si l'objet n'existe pas ou si la commande est appelée ailleurs que dans le contexte d'un formulaire, les coordonnées retournées sont (0;0;0;0).

Exemple

Vous souhaitez obtenir les coordonnées du rectangle formé par tous les objets dont le nom commence par "bouton" :

```
LIRE RECT OBJET(*;"bouton@";gauche;haut;droite;bas)
```

Référence

DEPLACER OBJET.

TAILLE OBJET OPTIMALE ({*; }objet; largeurOpti; hauteurOpti{; largeurMax})

Paramètre	Type		Description
*		→	Si spécifié = objet est un nom d'objet (chaîne) Si omis = objet est une variable
objet	Objet	→	Nom d'objet (si * est spécifié) ou Champ ou variable (si * est omis
largeurOpti	Entier long	←	Largeur optimale de l'objet
hauteurOpti	Entier long	←	Hauteur optimale de l'objet
largeurMax	Entier long	→	Largeur maximum de l'objet

Description

La commande TAILLE OBJET OPTIMALE retourne dans les paramètres largeurOpti et hauteurOpti la largeur et la hauteur "optimales" de l'objet de formulaire désigné par les paramètres * et objet. Ces valeurs sont exprimées en pixels. Cette commande est particulièrement utile dans le cadre de l'affichage ou de l'impression d'états complexes, associée à la commande DEPLACER OBJET.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne de caractères). Si vous ne passez pas le paramètre *, vous indiquez que objet est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (de type objet uniquement).

Les valeurs optimales retournées indiquent la taille minimale de l'objet pour que son contenu courant soit entièrement inclus dans ses limites. Bien entendu, ces valeurs n'ont de sens qu'avec des objets contenant du texte. Ce calcul tient compte de la police, de sa taille, de son style et du contenu de l'objet. Il tient compte également des césures et des retours chariot. Si l'objet spécifié est vide, la largeurOpti retournée est 0.

La taille retournée ne tient pas compte du cadre graphique éventuellement appliqué autour de l'objet ni des barres de défilement. Pour obtenir la taille réelle d'un objet à l'écran, il sera nécessaire d'ajouter l'épaisseur de ces éléments.

Le paramètre optionnel largeurMax vous permet d'attribuer une largeur maximale à l'objet. Si la largeur optimale de l'objet est supérieure à cette valeur, TAILLE OBJET OPTIMALE retourne largeurMax dans le paramètre largeurOpti et augmente la hauteur optimale en conséquence.

Les objets pris en charge par cette commande sont les suivants :
• Zones de texte statiques
• Textes insérés sous forme de références
• Champs et variables de type Alpha, Texte, Numérique, Entier, Entier long, Date, Heure, Booléens (cases à cocher et boutons radio)
• Boutons

Pour tous les autres types d'objets de formulaires (zones de groupes, onglets, rectangles, droites, cercles/ellipses, zones externes, etc.), la commande TAILLE OBJET OPTIMALE retourne la taille courante de l'objet (définie dans l'éditeur de formulaires et éventuellement à l'aide de la commande DEPLACER OBJET).

Exemple

Reportez-vous à l'exemple de la routine FIXER TAQUET IMPRESSION.

Référence

DEPLACER OBJET.

TITRE BOUTON ({*; }objet; libellBouton)

Paramètre	Type		Description
*		→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
libellBouton	Alpha	→	Nouveau nom du bouton

Description

La commande TITRE BOUTON change le titre du ou des bouton(s) spécifié(s) dans le paramètre objet et le remplace par la valeur définie dans le paramètre libellBouton.

Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objet est un nom d'objet (une chaîne). Si vous ne passez pas ce paramètre, vous indiquez que le paramètre objet est une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de variable (variable objet uniquement). Pour plus d'informations sur les noms d'objets, reportez-vous à la section Propriétés des objets.

TITRE BOUTON n'affecte que les boutons affichant du texte : boutons, cases à cocher et boutons radio.

Généralement, cette commande s'applique à un bouton à la fois. La zone du bouton doit être assez grande pour pouvoir accueillir le texte ; sinon, le texte est tronqué. N'utilisez pas de retours chariot dans libellBouton.

Exemple

L'exemple suivant est la méthode objet d'un bouton de recherche situé dans la zone de pied de page d'un formulaire sortie affiché par la commande MODIFIER SELECTION. La méthode effectue une recherche dans une table et active ou inactive le bouton intitulé bSuppr et change son titre, en fonction des résultats de la recherche :

```
CHERCHER ([Personnes]; [Personnes]Nom = vNom)
Au cas ou
   : (Enregistrements trouves ([Personnes]) = 0) ` Personne n'a été trouvé
      TITRE BOUTON (bSuppr;" Supprimer")
      INACTIVER BOUTON (bSuppr)
   : (Enregistrements trouves ([Personnes]) = 1) ` Une personne a été trouvée
      TITRE BOUTON (bSuppr;"Supprimer la personne")
      ACTIVER BOUTON (bSuppr)
   : (Enregistrements trouves([Personnes]) > 1) ` Plusieurs personnes ont été trouvées
      TITRE BOUTON (bSuppr;"Supprimer les personnes")
      ACTIVER BOUTON (bSuppr)
Fin de cas
```

Référence

ACTIVER BOUTON, INACTIVER BOUTON.

47

Protocole sécurisé

GENERER CLES CRYPTAGE

GENERER CLES CRYPTAGE (cléPrivée; cléPublique{; longueur})

Paramètre	Type		Description
cléPrivée	BLOB	←	BLOB devant recevoir la clé privée
cléPublique	BLOB	←	BLOB devant recevoir la clé publique
longueur	Entier long	→	Longueur des clés en bits [386...1024] Par défaut = 512

Description

La commande GENERER CLES CRYPTAGE génère une nouvelle paire de clés RSA. Ces clés, destinées au cryptage/décryptage des données, sont à la base du système de sécurisation des données proposé par 4D. Elles pourront être utilisées, dans le cadre du protocole SSL, avec le serveur Web 4D (cryptage et sécurité des communications) mais également dans toute base de données (cryptage de données).

Après l'exécution de la commande, les BLOB passés dans les paramètres cléPrivée et cléPublique contiennent une nouvelle paire de clés de cryptage.

Le paramètre optionnel longueur vous permet de préciser la taille (en bits) des clés que vous souhaitez obtenir. Plus une clé est longue, plus son décryptage "frauduleux" sera difficile. En contrepartie, plus les clés sont longues, plus les délais d'exécution ou de réponse seront importants, en particulier dans le cadre d'une connexion SSL.
Par défaut (si vous omettez le paramètre longueur), la taille des clés générée est de 512 bits. Vous pouvez générer des clés de 1024 bits, ce qui renforce la sécurité du cryptage, mais ralentira les connexions de votre application Web. Pour augmenter encore la sécurité, vous pouvez envisager de changer de paire de clés assez fréquemment, par exemple tous les six mois.

Note : Attention si vous générez des clés dans le but d'établir une demande de certificat SSL, seules les clés de 512 et de 1024 bits sont admises.

Les clés générées par cette commande sont au format standard PKCS, ce qui signifie que leur contenu peut être copié et collé dans un e-mail en toute sécurité et sans risque d'altération. Une fois que vous avez obtenu une paire de clés, vous pouvez générer un document texte (par exemple à l'aide de la commande BLOB VERS DOCUMENT) et stocker les clés dans un endroit sûr.

Important : La clé privée ne doit jamais être diffusée, sous quelque forme que ce soit.

RSA, clés privées et clés publiques

L'algorithme de cryptage RSA employé par la commande GENERER CLES CRYPTAGE est basé sur un système de cryptage à double clé : une clé privée et une clé publique. Comme son nom l'indique, la clé publique peut être diffusée auprès de tiers, et permet le décryptage des informations. Il lui correspond une clé privée unique, utilisée pour crypter les données. La clé privée sert au cryptage ; la clé publique, au décryptage (ou inversement). Ce qui est crypté avec une clé ne peut être décrypté qu'avec l'autre.
Les fonctions de cryptage du protocole SSL sont basées sur ce principe, la clé publique étant incluse dans le certificat envoyé aux navigateurs (cf. section Utiliser le protocole SSL).

Ce mode de cryptage est également utilisé par la première syntaxe des commande CRYPTER BLOB et DECRYPTER BLOB. Ce principe requiert que la clé publique soit diffusée de manière confidentielle.
Il est possible de mêler les clés publiques et privées de deux intervenants pour crypter des données de telle manière que seul le récepteur peut décrypter les données, et seul l'émetteur peut les avoir cryptées. C'est le principe de la seconde syntaxe des commandes CRYPTER BLOB et DECRYPTER BLOB.

Exemple

Reportez-vous à l'exemple de la commande CRYPTER BLOB.

Référence

CRYPTER BLOB, DECRYPTER BLOB, GENERER DEMANDE CERTIFICAT.

GENERER DEMANDE CERTIFICAT

version 6.7

GENERER DEMANDE CERTIFICAT (cléPrivée; demCertif; tabCodes; tabLibellés)

Paramètre	Type		Description
cléPrivée	BLOB	→	BLOB contenant la clé privée
demCertif	BLOB	←	BLOB devant recevoir la demande de certificat
tabCodes	Tab Entier long	→	Liste des codes d'informations
tabLibellés	Tab Alpha	→	Liste des libellés d'informations

Description

La commande GENERER DEMANDE CERTIFICAT permet de générer une demande de certificat au format PKCS, directement exploitable par des autorités de certification telles que Verisign® ou Thawthe®. Le certificat est une pièce essentielle du fonctionnement du protocole SSL dans le cadre d'un serveur Web. Il est envoyé à chaque browser se connectant en mode SSL. Il contient la "carte d'identité" du site Web (reprenant les informations que vous saisissez dans la commande), ainsi que sa clé publique — permettant aux browsers de décrypter les informations reçues. En outre, le certificat contient diverses informations ajoutées par l'autorité de certification.

Note : Pour plus d'informations sur le fonctionnement du protocole SSL avec le serveur Web 4D, reportez-vous à la section Utiliser le protocole SSL.

La demande de certificat nécessite une paire de clés générée à l'aide de la commande GENERER CLES CRYPTAGE et contient diverses informations. C'est en combinant cette demande avec d'autres paramètres qui lui sont propres, que l'autorité de certification sera en mesure de générer un certificat.

Passez dans cléPrivée un BLOB contenant la clé privée générée avec la commande GENERER CLES CRYPTAGE.

Passez dans demCertif un BLOB vide. Après l'exécution de la commande, il contiendra la demande de certificat au format PKCS. Vous pouvez placer cette demande dans un fichier texte, par exemple à l'aide de la commande BLOB VERS DOCUMENT, pour la faire parvenir à l'autorité de certification.

Important : La clé privée est utilisée pour générer la demande de certificat mais ne doit pas être envoyée à l'autorité de certification.

Vous devez remplir les tableaux tabCodes (de type entier long) et tabLibellés (de type alpha) avec, respectivement, les numéros de code et les libellés des informations destinées à l'autorité de certification.

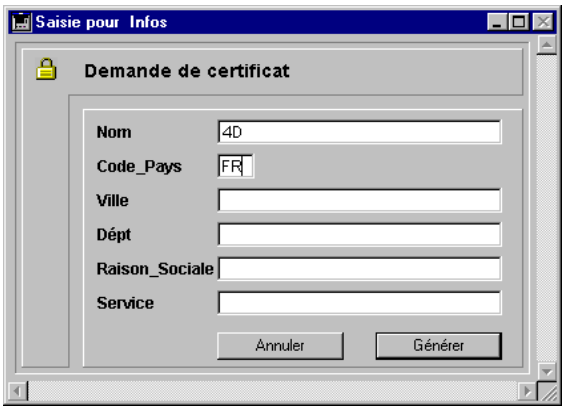
Les codes et les libellés attendus peuvent varier en fonction de l'autorité de certification et du mode d'utilisation du certificat. Toutefois, dans le cadre d'une utilisation standard du certificat (connexions d'un serveur Web via SSL), les tableaux doivent contenir les éléments suivants :

Informations à fournir (Exemples)	**tabCodes**	**tabLibellés**
CommonName : Nom du domaine	13	www.4D.fr
CountryName : Code du pays (deux lettres)	14	FR
LocalityName : Ville	15	Clichy
StateOrProvinceName : Département, Etat...	16	Hauts de Seine
OrganizationName : Raison sociale	17	4D
OrganizationUnit : Service/Personne en charge du serveur	18	Web Administrator

L'ordre dans lequel les codes et les informations sont insérés dans les tableaux n'a pas d'importance, en revanche les deux tableaux doivent être "synchronisés" : si l'élément {3} du tableau tabCodes contient la valeur *15* (nom de la ville), l'élément {3} du tableau tabLibellés doit contenir cette information, dans notre exemple *Clichy*.

Exemple

Un formulaire "Demande de certificat" comporte les six champs nécessaires à l'établissement d'une demande de certificat standard. Le bouton **Générer** crée un document sur disque contenant la demande de certificat. Le document "Cléprivée.txt" contient la clé privée (générée à l'aide la commande GENERER CLES CRYPTAGE) doit déjà être présent sur le disque.




```
` Méthode objet du bouton bGénérer
C_BLOB($vbcléPrivée;$vbDemandeCert)
C_ENTIER LONG($NumTable)
TABLEAU ENTIER LONG ($tLCodes;6)
TABLEAU ALPHA (80;$tAInfos;6)
```

```
$NumTable:=Table(Table du formulaire courant)
Boucle ($i;1;6)
    $tAInfos{$i}:= Champ($NumTable;$i)->
    $tLCodes{$i}:=$i+12
Fin de boucle
Si (Chercher dans tableau($tAInfos;"") # -1)
    ALERTE ("Vous devez remplir tous les champs.")
Sinon
    ALERTE ("Sélectionnez votre clé privée.")
    $vhRefDoc:=Ouvrir document("")
    Si (OK=1)
        FERMER DOCUMENT($vhRefDoc)
        DOCUMENT VERS BLOB(Document;$vbcléPrivée)
        GENERER DEMANDE CERTIFICAT($vbcléPrivée;$vbDemandeCert;$tLCodes;
                                                                    $tAInfos)
        BLOB VERS DOCUMENT ("Demande.txt";$vbDemandeCert)
    Sinon
        ALERTE ("Clé privée invalide.")
    Fin de si
Fin de si
```

Référence

GENERER CLES CRYPTAGE, Utiliser le protocole SSL.

48

Recherches et tris

CHERCHER ({table}{; critèreRecherche{; *}})

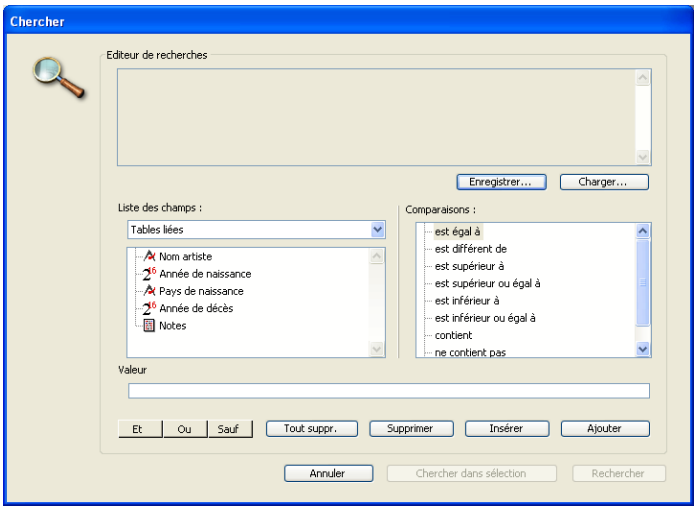
Paramètre	Type		Description
table	Table	→	Table dans laquelle la sélection est créée ou Table par défaut si ce paramètre est omis
critèreRecherche		→	Critère de recherche
*		→	Attente d'exécution de la recherche

Description

La commande CHERCHER recherche les enregistrements répondant au(x) critère(s) de recherche spécifié(s) dans critèreRecherche et retourne une sélection d'enregistrements de table. CHERCHER modifie la sélection courante de table pour le process courant. Le premier enregistrement de la nouvelle sélection devient l'enregistrement courant.

Si vous omettez le paramètre table, la commande s'applique à la table par défaut. Si aucune table par défaut n'a été définie, une erreur est générée.

Si vous ne passez ni le paramètre critèreRecherche ni le paramètre *, CHERCHER affiche la boîte de dialogue de l'Editeur de recherches de 4D pour table (sauf lorsqu'il s'agit de la dernière ligne d'une recherche complexe, cf. ci-dessous) :



Pour plus d'informations sur l'utilisation de cet éditeur, reportez-vous au manuel *Mode Développement*.
L'utilisateur construit la recherche puis clique sur le bouton **Rechercher** ou **Chercher dans sélection**. Si la recherche est correctement effectuée et n'est pas interrompue, la variable système OK prend la valeur 1. Si l'utilisateur clique sur **Annuler**, la commande CHERCHER est interrompue sans effectuer de recherche et la variable OK prend la valeur 0 (zéro).

Exemples

(1) L'exemple suivant affiche l'Editeur de recherches pour la table [Produits] :

```
CHERCHER([Produits])
```

(2) L'exemple suivant affiche l'Editeur de recherches pour la table par défaut (si elle a été définie) :

```
CHERCHER
```

Si vous spécifiez le paramètre critèreRecherche, l'Editeur de recherches ne s'affiche pas et la recherche est entièrement définie par programmation. Pour des recherches simples (recherches sur un seul champ), vous appelez CHERCHER une seule fois avec le paramètre critèreRecherche construit de la manière décrite plus bas. Pour des recherches complexes (recherches sur de multiples champs ou avec de multiples conditions), vous appelez CHERCHER autant de fois que nécessaire avec le paramètre critèreRecherche et le paramètre optionnel * sauf pour la dernière ligne CHERCHER (qui déclenche la recherche).

Exemples

(3) L'exemple suivant recherche les [Personnes] dont le nom commence par "a" :

```
CHERCHER([Personnes];[Personnes]Nom="a@")
```

(4) L'exemple suivant recherche les [Personnes] dont le nom commence par "a" ou "b" :

```
CHERCHER([Personnes];[Personnes]Nom="a@";*)
   ` * indique qu'il y a un autre critère de recherche
CHERCHER([Personnes]; |;[Personnes]Nom="b@")
   ` Pas de * : indique la fin de la définition des critères et lance l'exécution de la recherche
```

Note : Le mode d'interprétation du caractère @ dans les recherches peut être modifié via une option des préférences. Pour plus d'informations, reportez-vous à la section Opérateurs de comparaison.

Construction d'une ligne de recherche

Le paramètre critèreRecherche utilise la syntaxe suivante :

{opérateur ; } champ comparateur valeur

• L'opérateur est utilisé pour lier deux appels à CHERCHER lors d'une définition de recherche complexe. Les opérateurs disponibles sont les mêmes que ceux proposés dans l'Editeur de recherches :

Opérateur	Symbole
ET	&
OU	|
Sauf	#

L'opérateur est optionnel et n'est pas nécessaire pour le premier appel à CHERCHER lors d'une recherche complexe. Il est également inutile si votre recherche s'écrit sur une seule ligne. Si vous l'omettez à l'intérieur d'une recherche complexe, le ET (&) est utilisé par défaut.

• Le champ est le champ sur lequel va porter la recherche. Il peut provenir d'une autre table si celle-ci est la table 1 d'une table liée à table par un lien automatique ou manuel.

• Le comparateur est l'élément qui va permettre de confronter champ et critèreRecherche. Voici la liste des comparateurs possibles :

Comparateur	Symbole à utiliser avec CHERCHER
Egal à	=
Différent de	#
Inférieur à	<
Supérieur à	>
Inférieur ou égal à	<=
Supérieur ou égal à	>=
Contient mot-clé	%

Note : Il est possible de définir le comparateur sous la forme d'une expression alphanumérique au lieu d'un symbole. Dans ce cas, il est obligatoire d'utiliser des points-virgules pour dissocier les éléments de la chaîne de recherche. Ce principe permet par exemple de créer des séquences de recherches paramétrables en faisant varier le comparateur, ou de construire des interfaces de recherche utilisateur personnalisées. Reportez-vous à l'exemple 19.

• La valeur représente ce qui va être confronté au contenu de champ. La valeur peut être toute expression du même type que champ. Le type de la valeur n'est évalué qu'une seule fois, au démarrage de la recherche, et ne l'est donc pas pour chaque enregistrement. Si la recherche porte sur le contenu d'une chaîne de caractères, utilisez dans 'valeur' le symbole "@" pour isoler le contenu à rechercher, par exemple "@Dupon@". Il est à noter, dans ce cas, que vous ne bénéficierez pas d'une recherche indexée.
La recherche par mots-clés n'est disponible qu'avec des champs de type alpha ou texte. Pour plus d'informations sur ce type de recherche, reportez-vous à la section Opérateurs de comparaison.

Voici les règles à observer pour la construction de séquences de recherche :
• La première ligne ne doit pas contenir d'opérateur.
• Les suivantes peuvent débuter par un opérateur de liaison. Si vous l'omettez, l'opérateur ET (&) est utilisé par défaut.
• Toutes les lignes, à l'exception de la dernière, doivent s'achever par le symbole *.
• Pour lancer la recherche, ne passez pas le paramètre * lors de la construction de votre dernière ligne. Autre solution : vous pouvez exécuter la commande CHERCHER sans autre paramètre que la table (l'Editeur de recherches ne s'affiche pas ; au lieu de cela, les lignes de recherche complexes définies auparavant sont exécutées).

Note : Chaque table maintient sa propre construction de recherche courante. Cela signifie que vous pouvez créer de multiples recherches simultanément, une pour chaque table. Dans ce cas, vous devez passer le paramètre table ou spécifier une table par défaut.

Quelle que soit la manière dont la recherche a été définie :

• Si l'exécution d'une commande CHERCHER nécessite un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Ces thermomètres peuvent être cachés à l'aide des commandes SUPPRIMER MESSAGES et LAISSER MESSAGES. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton **Stop** pour interrompre l'opération. Si la recherche s'est correctement déroulée, la variable système OK prend la valeur 1. Sinon, si la recherche est interrompue, OK prend la valeur 0 (zéro).

• Si des champs indexés sont spécifiés, la recherche est optimisée à chaque fois que c'est possible (la recherche commence par les champs indexés), réduisant au maximum la durée de l'opération. La commande tire parti des index composites pour les recherches utilisant le ET (&).

Exemples

(5) Nous recherchons tous les enregistrements dont le nom correspond à "Dupont" :

```
CHERCHER([Personnes]; [Personnes]Nom = "Dupont")
```

Note : Si le champ Nom est indexé, nous bénéficions donc d'une recherche accélérée tirant parti de l'index.
Rappel : Cette recherche trouvera les enregistrements tels que "Dupont", "dupont", "DUPONT", etc. Si vous voulez que la recherche tienne compte des majuscules/minuscules, définissez des critères supplémentaires utilisant les codes de caractères.

(6) Nous recherchons les personnes se nommant "Dupont" et se prénommant "Jean". Le champ Nom est indexé. En revanche, le champ Prénom ne l'est pas :

```
CHERCHER ([Personnes]; [Personnes]Nom = "Dupont"; *)
   ` Chercher toute personne qui s'appelle Dupont
CHERCHER ([Personnes]; &;[Personnes]Prénom = "Jean")  ` dont le prénom est Jean
```

Cet exemple effectue dans un premier temps une recherche rapide sur le champ indexé Nom, ce qui réduit la sélection d'enregistrements à ceux des personnes s'appelant Dupont. La recherche s'effectue ensuite séquentiellement sur le champ Prénom, mais nous serons peu pénalisés puisqu'elle s'exécute parmi une présélection d'enregistrements.

Note : Cette recherche est particulièrement optimisée si la base contient un index composite incluant les champs [Personnes]Nom+[Personnes]Prénom. Dans ce cas, la commande tire parti de l'index et la recherche est entièrement indexée.

(7) L'exemple suivant recherche les personnes se nommant Dupont ou Blanc. Le champ Nom est indexé :

```
   ` Chercher toute personne qui s'appelle Dupont…
CHERCHER ([Personnes]; [Personnes]Nom = "Dupont"; *)
CHERCHER ([Personnes]; |;[Personnes]Nom = "Blanc")  ` ou Blanc
```

La commande utilise l'index du champ Nom pour les deux recherches. Les deux recherches sont effectuées, et leurs résultats sont placés dans des ensembles internes qui sont finalement combinés par l'intermédiaire d'une opération Union.

(8) L'exemple suivant recherche des personnes qui ne travaillent pas pour une société. La recherche est effectuée en testant si le nom de la société est une chaîne vide.

```
CHERCHER ([Personnes]; [Personnes]Société = "") ` Chercher les personnes sans société
```

(9) L'exemple suivant recherche chaque personne se nommant "Dupont" et travaillant dans une société basée à Paris. La deuxième recherche utilise un champ venant d'une autre table. Cette recherche peut être effectuée parce que la table [Personnes] est liée à la table [Société] par un lien de N vers 1 :

```
CHERCHER ([Personnes]; [Personnes]Nom = "Dupont"; *)
    ` Chercher toute personne qui s'appelle Dupont…
CHERCHER ([Personnes]; &;[Société]Ville = "Paris")  ` ...qui travaille pour une société à Paris
```

(10) L'exemple suivant recherche l'enregistrement de chaque personne dont l'initiale du nom est située entre les lettre A (incluse) et M (incluse) :

```
CHERCHER ([Personnes]; [Personnes]Nom < "n")  ` Trouver toute personne entre A et M
```

(11) L'exemple suivant recherche les enregistrements des personnes habitant soit Paris soit Lyon :

```
CHERCHER ([Personnes]; [Personnes]CodePostal = "75@"; *)
    ` Trouver ceux qui habitent Paris…
CHERCHER ([Personnes]; |;[Personnes]CodePostal = "6900@")  ` ou Lyon
```

(12) Recherche par mot-clé : l'exemple suivant recherche dans toute la table [Produits] les enregistrements dont le champ Description contient le mot "facile" :

```
CHERCHER([Produits];[Produits]Description%"facile")
    ` Trouver les produits dont la description contient le mot-clé facile
```

(13) Nous recherchons les enregistrements correspondant à la réponse fournie dans une boîte de dialogue :

```
vTrouvé := Demander ("Saisissez un code de facture :")
    `Demander un code de facture à l'utilisateur
Si (OK = 1)  ` Si l'utilisateur clique sur OK…
    CHERCHER ([Factures]; [Factures]Code = vTrouvé)
        `Trouver le code qui correspond à vTrouvé
Fin de si
```

(14) Cet exemple recherche tous les enregistrements des factures saisies en 1996. Nous recherchons les dates entre le 31/12/95 et le 1/1/97 :

```
CHERCHER ([Factures]; [Factures]DateFacture > !31/12/95!; *)
    ` Trouver des factures après le 31/12/95…
CHERCHER ([Factures]; &;[Factures]DateFacture < !1/1/97!) ` et avant 1/1/97
```

(15) L'exemple suivant trouve les employés qui ont un salaire entre 20 000 et 40 000 Euros. La recherche inclut les employés qui gagnent 20 000 Euros et exclut ceux qui gagnent 40 000 Euros :

```
CHERCHER ([Employés]; [Employés]Salaire >= 20000; *)
    ` Trouver les employés qui ont un salaire entre…
CHERCHER ([Employés]; &;[Employés]Salaire < 40000)  ` 20 000 et 40 000 Euros
```

(16) L'exemple suivant cherche les employés du service Marketing qui ont un salaire supérieur à 30 000 Euros. Le champ Salaire est utilisé dans un premier temps car il est indexé. Notez que la seconde recherche utilise un champ venant d'une autre table. Le champ [Service]Nom est lié à la table [Employés] par un lien automatique de N vers 1.

```
CHERCHER ([Employés]; [Employés]Salaire > 30000; *)
    ` Trouver les employés qui ont un salaire supérieur à 30 000 Euros
CHERCHER ([Employés]; &;[Service]Nom = "marketing")
    ` et qui travaillent dans le service marketing
```

(17) Soient trois tables reliées par des liens de N vers 1 : [Ville] -> [Département] -> [Région] . La recherche suivante trouve toutes les régions comportant des villes dont le nom débute par "Saint" :

```
CHERCHER ([Région]; [Ville]Nom="Saint@")
    ` Trouver toutes les régions contenant des villes commençant par Saint
```

(18) La recherche suivante recherche les informations égales à la valeur de la variable mavar.

```
CHERCHER ([Lois]; [Lois]Texte = mavar)
    ` Trouver toutes les lois qui sont égales à la valeur de mavar
```

La recherche peut avoir des résultats différents selon la valeur de mavar. Elle sera également exécutée différement. Par exemple :
• Si mavar est égale à "Copyright@", la sélection contient toutes les lois qui commencent par Copyright.
• Si mavar est égale à "@Copyright@", la sélection contient toutes les lois qui contiennent au moins une occurrence de Copyright.

(19) L'exemple suivant ajoute ou non les lignes d'une recherche complexe en fonction de la valeur de variables. Ainsi, seuls les critères valides sont pris en compte pour la recherche :

```
CHERCHER([Facture];[Facture]Payee=Faux;*)
Si($ville#"") ` Si un nom de ville a été spécifié
    CHERCHER([Facture];[Facture]Ville_Livraison=$ville;*)
Fin de si
Si ($code_postal#"") ` Si un code postal a été spécifié
    CHERCHER([Facture];[Facture]Code_Postal=$code_postal;*)
Fin de si
CHERCHER([Facture]) ` Exécution de la recherche sur les critères
```

(20) Cet exemple illustre l'utilisation d'un comparateur sous forme d'expression alphanumérique. La valeur du comparateur est définie via un pop up menu placé dans une boîte dialogue de recherche personnalisée :

```
C_TEXTE($ope)
$ope:=_pup_operateur{_pup_operateur} `$ope vaut par exemple "#", ou "="
Si (OK =1)
    CHERCHER([Facture];[Facture]Montant;$ope;$montant)
Fin de si
```

Référence

CHERCHER DANS SELECTION, Opérateurs.

Variables et ensembles système

Si la recherche est correctement effectuée, la variable système OK prend la valeur 1.
La variable OK prend la valeur 0 si :
- l'utilisateur clique sur **Annuler** dans la boîte de dialogue de recherche,
- en mode 'recherche et verrouillage' (cf. commande FIXER RECHERCHE ET VERROUILLAGE), la recherche a trouvé au moins un enregistrement verrouillé. Dans ce cas également, l'ensemble système LockedSet est mis à jour.

CHERCHER DANS SELECTION

version 6.5 (Modifiée)

CHERCHER DANS SELECTION ({laTable}{; critère{; *}})

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle effectuer la recherche ou ou Table par défaut si ce paramètre est omis
critère		→	Lignes de recherche
*		→	Attente d'exécution de la recherche

Description

CHERCHER DANS SELECTION recherche des enregistrements dans laTable. CHERCHER DANS SELECTION modifie la sélection courante de laTable pour le process courant. Le premier enregistrement de la nouvelle sélection devient l'enregistrement courant.

CHERCHER DANS SELECTION a un fonctionnement et des effets proches de ceux de CHERCHER. La différence entre ces deux commandes est la portée de la recherche :
• CHERCHER recherche des enregistrements dans la table.
• CHERCHER DANS SELECTION recherche des enregistrements parmi la sélection courante de la table.

Pour plus d'informations, reportez-vous à la description de la commande CHERCHER.

Exemple

Cet exemple illustre la différence entre CHERCHER et CHERCHER DANS SELECTION. Voici deux recherches :

```
      ` Trouver TOUTES les sociétés basées à Paris
CHERCHER ([Sociétés]; [Sociétés]Ville="Paris")
      ` Trouver TOUTES les sociétés s'occupant d'affaires boursières (quelle que soit leur ville)
CHERCHER ([Sociétés]; [Sociétés]Activité="Affaires boursières")
```

Notez que le second CHERCHER "ignore" complètement les résultats du premier. Comparez avec :

```
      ` Trouver TOUTES les sociétés basées à Paris
CHERCHER ([Sociétés]; [Sociétés]Ville="Paris")
      ` Trouver TOUTES les sociétés s'occupant d'affaires boursières basées à Paris
CHERCHER DANS SELECTION ([Sociétés]; [Sociétés]Activité="Affaires boursières")
```

CHERCHER DANS SELECTION n'effectue sa recherche que parmi les enregistrements sélectionnés, dans cet exemple les sociétés basées à Paris.

Référence

CHERCHER.

CHERCHER PAR EXEMPLE ({table}{; }{*})

Paramètre	Type		Description
table	Table	→	Table de laquelle une sélection d'enregistrements doit être retournée ou Table par défaut si ce paramètre est omis
*	*	→	Masquer les barres de défilement

Description

La commande CHERCHER PAR EXEMPLE effectue la même action que la commande de menu **Recherche par formulaire**... en mode Développement. Cette commande affiche le formulaire entrée courant comme fenêtre de recherche. CHERCHER PAR EXEMPLE cherche dans table les données que l'utilisateur a saisies dans cette fenêtre. Le formulaire doit contenir les champs sur lesquels vous voulez que l'utilisateur puisse effectuer la recherche. La recherche est optimisée : les champs indexés sont automatiquement utilisés.
Si vous passez le paramètre optionnel *, les barres de défilement du formulaire sont masquées.

Reportez-vous au manuel *Mode Développement* de 4D pour plus d'informations sur l'utilisation de la commande de menu **Recherche par formulaire...** du mode Développement.

Exemple

La méthode dans l'exemple suivant affiche le formulaire maRecherche. Si l'utilisateur valide le formulaire et exécute la recherche (c'est-à-dire si la variable système OK prend la valeur 1), les enregistrements trouvés sont affichés :

```
FORMULAIRE ENTREE ([Personnes]; "maRecherche")
   ` Ce formulaire devient le formulaire entrée
CHERCHER PAR EXEMPLE ([Personnes]) ` Afficher le formulaire pour la recherche
Si (OK = 1) ` Si l'utilisateur valide la recherche
   VISUALISER SELECTION ([Personnes]) ` Visualiser les enregistrements trouvés
Fin de si
```

Référence

CHERCHER, TRIER.

Variables et ensembles système

Si l'utilisateur clique sur le bouton Valider ou appuie sur la touche Entrée, la variable système OK prend la valeur 1 et la recherche est effectuée. Si l'utilisateur clique sur le bouton Annuler ou utilise la touche d'annulation, la variable système OK prend la valeur 0 et la recherche est annulée.

CHERCHER PAR FORMULE

version 11.2 (Modifiée)

CHERCHER PAR FORMULE (laTable{; formule})

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle effectuer la recherche
formule	Booléen	→	Formule de recherche

Description

CHERCHER PAR FORMULE effectue une recherche d'enregistrements dans laTable. CHERCHER PAR FORMULE modifie la sélection courante de laTable pour le process courant et fait du premier enregistrement le nouvel enregistrement courant.

CHERCHER PAR FORMULE et la commande CHERCHER PAR FORMULE DANS SELECTION fonctionnent exactement de la même manière, à la différence près que CHERCHER PAR FORMULE effectue sa recherche parmi la totalité des enregistrements de la table alors que CHERCHER PAR FORMULE DANS SELECTION se cantonne aux enregistrements de la sélection courante.

Les deux commandes appliquent formule à chaque enregistrement de la table ou de la sélection. formule est une expression booléenne qui doit retourner VRAI ou FAUX. Si formule retourne Vrai, l'enregistrement est inclus dans la nouvelle sélection.

formule peut être simple (par exemple la comparaison d'un champ à une valeur) ou complexe (réalisation d'un calcul ou même évaluation de valeurs dans une table liée). Ce peut être une fonction 4D, ou une fonction ou une expression que vous avez créée. Lorsque vous travaillez avec des champs de type Alpha ou Texte, vous pouvez utiliser dans formule des jokers (@) ainsi que l'opérateur "contient" (%) pour la recherche par mots-clés. Pour plus d'informations, reportez-vous à la description de la commande CHERCHER.

Si vous omettez le paramètre formule, 4D affiche la boîte de dialogue standard de recherche.

Lorsque la recherche est terminée, le premier enregistrement de la nouvelle sélection est chargé depuis le disque et devient l'enregistrement courant.

Ces commandes sont optimisées et peuvent notamment tirer parti des index. Lorsque le type de requête le permet, ces commandes exécutent des requêtes équivalentes à un CHERCHER. Par exemple, l'instruction CHERCHER PAR FORMULE([matable]; [matable]monchamp=valeur) sera exécutée comme CHERCHER([matable]; [matable]monchamp=valeur), ce qui permettra d'utiliser l'index. 4D pourra également optimiser les requêtes contenant des parties non optimisables, en exécutant d'abord les parties optimisables puis en combinant les résultats avec le reste de la requête. Par exemple, l'instruction CHERCHER PAR FORMULE([matable];Longueur(monchamp)=valeur) ne sera pas optimisée. En revanche, CHERCHER PAR FORMULE([matable];Longueur(monchamp)=valeur1 | monchamp=valeur2) sera partiellement optimisée.

Ces commandes effectuent par défaut des "jointures" à la manière du SQL. Avec ce principe, les recherches sont optimisées et il n'est pas nécessaire qu'un lien automatique structurel existe entre la table A et la table B pour pouvoir exécuter une instruction du type CHERCHER PAR FORMULE([Table_A];[Table_A]champ_X = [Table_B]champ_Y) (cf. exemple 3).
S'ils existent, les liens définis dans l'éditeur de structure ne sont en principe pas utilisés. Toutefois, ces commandes utilisent les liens automatiques dans les cas suivants :
- si la formule ne peut se décomposer en éléments de la forme { champ ; comparateur ; valeur}
- si deux champs de la même table sont comparés.

Note : Pour des raisons de compatibilité, il est possible de désactiver le mécanisme de jointures, soit globalement via les Préférences de la base (bases de données converties uniquement) soit par process via la commande FIXER PARAMETRE BASE.

4D Server : A compter de la version 11 de 4D Server, ces commandes sont exécutées sur le serveur, ce qui optimise leur exécution. A noter qu'en cas d'appel direct de variables dans la formule, la requête est calculée avec la valeur de la variable sur le poste client. Par exemple, l'instruction CHERCHER PAR FORMULE([matable];[matable]monchamp=mavariable) sera exécutée sur le serveur mais avec le contenu de la variable *mavariable* du client.
En revanche, ce principe n'est pas appliqué pour les formules utilisant des méthodes qui, elles-mêmes, font appel à des variables (la valeur des variables est évaluée sur le serveur). Dans ce contexte, il peut être judicieux d'utiliser l'attribut de méthode "Exécuter sur serveur" permettant d'exécuter une méthode sur le serveur en lui passant des paramètres (variables) (cf. manuel *Mode Développement*).
Dans les versions précédentes de 4D Server, ces commandes étaient exécutées sur les postes clients. Par compatibilité, ce fonctionnement est maintenu dans les bases de données converties en version 11. Une préférence de compatibilité et un sélecteur de la commande FIXER PARAMETRE BASE permettent toutefois d'adopter le fonctionnement de la version 11 (exécution sur le serveur) dans les bases de données converties.

Exemples

(1) L'exemple suivant recherche les enregistrements de toutes les factures qui ont été saisies au mois de décembre, sans tenir compte de l'année. Le principe est d'appliquer la fonction Mois de à chaque enregistrement. Cette recherche ne pourrait pas être effectuée d'une autre manière sans créer un champ séparé pour le mois :

```
CHERCHER PAR FORMULE ([Factures]; Mois de ([Factures]Saisie) = 12)
    ` Chercher les factures saisies en décembre
```

(2) L'exemple suivant recherche les enregistrements de toutes les personnes dont le nom comporte plus de dix caractères :

```
CHERCHER PAR FORMULE ([Personnes]; Longueur ([Personnes]Nom) > 10)
    ` Chercher les personnes dont le nom fait plus de dix caractères
```

(3) Cet exemple active les jointures SQL pour une recherche par formule spécifique :

```
$valcourante:= Lire parametre base(Jointures CHERCHER PAR FORMULE)
FIXER PARAMETRE BASE(Jointures CHERCHER PAR FORMULE;2) `Activer les jointures SQL
    `Chercher toutes les lignes de factures du client "ACME" alors que les tables ne sont pas
    ` liées
CHERCHER PAR FORMULE([ligne_factures] ; ([ligne_factures]facture_id = [facture]id) &
                                                    ([facture]client = "ACME"))
FIXER PARAMETRE BASE(Jointures CHERCHER PAR FORMULE;$valcourante)
    `On rétablit le paramétrage courant
```

Référence

CHERCHER, CHERCHER DANS SELECTION, CHERCHER PAR FORMULE DANS SELECTION, CHERCHER PAR SQL.

CHERCHER PAR FORMULE DANS SELECTION

version 11.2 (Modifiée)

CHERCHER PAR FORMULE DANS SELECTION (laTable{; formule})

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle effectuer la recherche parmi la sélection courante
formule	Booléen	→	Formule de recherche

Description

La commande CHERCHER PAR FORMULE DANS SELECTION vous permet de rechercher des enregistrements dans laTable. CHERCHER PAR FORMULE DANS SELECTION modifie la sélection courante de laTable pour le process courant et fait du premier enregistrement le nouvel enregistrement courant.

CHERCHER PAR FORMULE DANS SELECTION fonctionne de la même manière que CHERCHER PAR FORMULE. La différence entre ces deux commandes se situe au niveau de la portée de la recherche :
• CHERCHER PAR FORMULE effectue sa recherche parmi la totalité des enregistrements de la table.
• CHERCHER PAR FORMULE DANS SELECTION effectue sa recherche uniquement parmi les enregistrements de la sélection courante.

Pour plus d'informations, reportez-vous à la description de la commande CHERCHER PAR FORMULE.

Référence

CHERCHER, CHERCHER DANS SELECTION, CHERCHER PAR FORMULE.

CHERCHER PAR TABLEAU

CHERCHER PAR TABLEAU (champCible; tableau)

Paramètre	Type		Description
champCible	Champ	→	Champ duquel comparer les valeurs
tableau	Tableau	→	Tableau des valeurs recherchées

Description

La commande CHERCHER PAR TABLEAU recherche dans la table du champ passé en premier paramètre tous les enregistrements pour lesquels la valeur de champCible est égale à au moins une des valeurs des éléments du tableau tableau. Les enregistrements trouvés constituent la nouvelle sélection courante.

Cette commande permet de construire rapidement et simplement une recherche sur plusieurs valeurs.

Notes :
• Cette commande ne peut pas être utilisée avec des champs de type image, sous-table et BLOB.
• champCible et tableau doivent impérativement être du même type. Exception : vous pouvez utiliser un tableau de type Entier long avec un champ de type Heure.

Exemple

Cet exemple permet de récupérer les enregistrements des clients français et américains :

```
TABLEAU ALPHA (2;TabRecherche;2)
TabRecherche{1}:="FR"
TabRecherche{2}:="US"
CHERCHER PAR TABLEAU ([Clients]Pays;TabRecherche)
```

CHERCHER PAR TABLEAU DANS SELECTION (champCible; leTableau)

Paramètre	Type		Description
champCible	Champ	→	Champ duquel comparer les valeurs
leTableau	Tableau	→	Tableau des valeurs recherchées

Description

La commande CHERCHER PAR TABLEAU DANS SELECTION recherche dans la table du champ passé en premier paramètre les enregistrements pour lesquels la valeur de champCible est égale à au moins une des valeurs des éléments du tableau leTableau. Les enregistrements trouvés constituent la nouvelle sélection courante.

CHERCHER PAR TABLEAU DANS SELECTION fonctionne de la même manière que CHERCHER PAR TABLEAU. La différence entre ces deux commandes se situe au niveau de la portée de la recherche :
• CHERCHER PAR TABLEAU effectue sa recherche parmi la totalité des enregistrements de la table de champCible.
• CHERCHER PAR TABLEAU DANS SELECTION effectue sa recherche uniquement parmi les enregistrements de la sélection courante de la table de champCible.

Pour plus d'informations, reportez-vous à la description de la commande CHERCHER PAR TABLEAU.

Référence

CHERCHER PAR TABLEAU.

DECRIRE EXECUTION RECHERCHE (statut)

Paramètre	Type		Description
statut	Booléen	→	Vrai=Enregistrer la description des requêtes, Faux=Stopper l'enregistrement

Description

La commande DECRIRE EXECUTION RECHERCHE permet d'activer ou d'inactiver le mode d'analyse de l'exécution des recherches pour le process courant. La commande tient compte indifféremment des recherches effectuées via le langage 4D ou des requêtes SQL.

L'appel de la commande avec le paramètre statut à Vrai active le mode d'analyse des recherches. Dans ce mode, le moteur de 4D enregistrera en interne deux séries d'informations spécifiques lors de chaque requête effectuée par la suite sur les données :
• la description détaillée de la recherche juste avant son exécution, c'est-à-dire la recherche prévue (le plan de recherche),
• la description détaillée de la recherche telle qu'elle a réellement été exécutée (le chemin de recherche).

Les informations enregistrées incluent le type de recherche (indexée, séquentielle), le nombre d'enregistrements trouvés et le temps nécessaire à l'exécution de chaque critère de recherche. Vous pouvez ensuite lire ces informations à l'aide des commandes Lire dernier plan recherche et Lire dernier chemin recherche.

En général, la description du plan d'une recherche et celle de son chemin sont identiques, mais elles peuvent toutefois différer car 4D peut mettre en oeuvre des optimisations dynamiques au cours de l'exécution de la recherche, dans le but d'améliorer les performances. Par exemple, une recherche indexée peut être convertie dynamiquement en recherche séquentielle si le moteur de 4D estime qu'elle sera plus rapide — c'est le cas notamment lorsque le nombre d'enregistrements parmi lesquels effectuer la recherche est faible.

Passez Faux dans le paramètre statut lorsque vous n'avez plus besoin d'analyser les recherches. Le mode d'analyse de l'exécution des recherches peut ralentir l'application.

Exemple

L'exemple suivant illustre le type d'information obtenue via ces commandes en cas de requête SQL :

```
C_TEXTE($vResultPlan;$vResultPath)
TABLEAU TEXTE(aTitles;0)
```

```
TABLEAU TEXTE(aDirectors;0)
DECRIRE EXECUTION RECHERCHE(Vrai) `mode analyse
Debut SQL
   SELECT ACTORS.FirstName, CITIES.City_Name
      FROM ACTORS, CITIES
      WHERE ACTORS.Birth_City_ID=CITIES.City_ID
      ORDER BY 1
      INTO :aTitles, :aDirectors;
Fin SQL
$vResultPlan:=Lire dernier plan recherche(Description format Texte)
$vResultPath:=Lire dernier chemin recherche(Description format Texte)
DECRIRE EXECUTION RECHERCHE(Faux)  `Fin du mode analyse
```

A l'issue de l'exécution de ce code, $vResultPlan et $vResultPath contiennent les descriptions des recherches effectuées, par exemple :

```
$vResultPlan :
   [Join] : ACTORS.Birth_City_ID = CITIES.City_ID
$vResultPath :
   And
      [Merge] : ACTORS with CITIES
         [Join] : ACTORS.Birth_City_ID = CITIES.City_ID  (1227 records found in 13 ms)
      → 1227 records found in 13 ms
   → 1227 records found in 14 ms
```

Si la constante Description format XML est passée à la commande Lire dernier chemin recherche, $vResultPath contient la description de la recherche exprimée en XML :

```
$vResultPath :
   <QueryExecution>
      <steps description="And" time="0" recordsfounds="1227">
         <steps description="[Merge] : ACTORS with CITIES" time="13"
recordsfounds="1227">
            <steps description="[Join] : ACTORS.Birth_City_ID =CITIES.City_ID"
time="13" recordsfounds="1227"/>
         </steps>
      </steps>
   </QueryExecution>
```

Référence

Lire dernier chemin recherche, Lire dernier plan recherche.

FIXER DESTINATION RECHERCHE

version 11 (Modifiée)

FIXER DESTINATION RECHERCHE (destinationType{; destinationObjet})

Paramètre	Type		Description
destinationType	Numérique	→	0 sélection courante 1 ensemble 2 sélection temporaire 3 variable
destinationObjet	Chaîne | Variable	→	Nom de l'ensemble ou Nom de la sélection temporaire ou Variable

Description

La commande FIXER DESTINATION RECHERCHE vous permet d'indiquer à 4D où placer les résultats de toutes les recherches qui suivent l'appel de cette commande dans le process courant.

Vous spécifiez le type de la destination dans le paramètre destinationType. 4D fournit les constantes prédéfinies suivantes, placées dans le thème "Recherches" :

Constante	Type	Valeur
Vers sélection courante	Entier long	0
Vers ensemble	Entier long	1
Vers sélection temporaire	Entier long	2
Vers variable	Entier long	3

Vous spécifiez le nom de la destination de la recherche dans le paramètre optionnel destinationObjet en fonction du tableau suivant :

Paramètre destinationType	Paramètre destinationObjet
0 (sélection courante)	Vous ne passez pas de paramètre.
1 (ensemble)	Vous passez le nom de l'ensemble (existant ou à créer)
2 (sélection temporaire)	Vous passez le nom de la sélection temporaire (existante ou à créer)
3 (variable)	Vous passez une variable numérique (existante ou à créer)

Avec

FIXER DESTINATION RECHERCHE(Vers sélection courante)

Les enregistrements trouvés par la recherche seront placés dans la sélection courante de la table dans laquelle la recherche est effectuée.

Avec

FIXER DESTINATION RECHERCHE(Vers ensemble;"monEnsem")

Les enregistrements trouvés par la recherche seront placés dans l'ensemble monEnsem. La sélection courante et l'enregistrement courant de la table dans laquelle vous recherchez restent inchangés.

Note : En client-serveur, vous ne pouvez pas utiliser comme destination de recherche un ensemble local/client (dont le nom est précédé du symbole $). En effet, ce type d'ensemble est stocké sur les postes clients alors que les recherches sont effectuées sur le serveur. Pour plus d'informations sur les types d'ensembles, reportez-vous à la section Présentation des ensembles.

Avec

FIXER DESTINATION RECHERCHE(Vers sélection temporaire;"maTemp")

Les enregistrements trouvés par la recherche seront placés dans la sélection temporaire maTemp. La sélection courante et l'enregistrement courant pour la table sur laquelle vous effectuez la recherche restent inchangés.

Note : Si la sélection temporaire n'existe pas avant l'appel, elle est automatiquement créée une fois la recherche effectuée.

Avec

FIXER DESTINATION RECHERCHE(Vers variable;$vlRésultatRech)

Le **nombre** d'enregistrements trouvés par la recherche sera placé dans la variable $vlRésultatRech. La sélection courante et l'enregistrement courant de la table dans laquelle vous effectuez la recherche restent inchangés.

Attention : FIXER DESTINATION RECHERCHE affecte toutes les recherches suivantes dans le process courant. N'oubliez pas d'associer toujours un appel à FIXER DESTINATION RECHERCHE (lorsque destinationType#0) à un appel à FIXER DESTINATION RECHERCHE(0) ultérieur pour rétablir le mode standard de recherche.

FIXER DESTINATION RECHERCHE modifie uniquement le comportement des commandes de recherche, c'est-à-dire :

- CHERCHER
- CHERCHER DANS SELECTION
- CHERCHER PAR EXEMPLE
- CHERCHER PAR FORMULE
- CHERCHER PAR FORMULE DANS SELECTION
- CHERCHER PAR SQL
- CHERCHER PAR TABLEAU
- CHERCHER PAR TABLEAU DANS SELECTION

En revanche, FIXER DESTINATION RECHERCHE n'affecte pas les autres commandes qui modifient la sélection courante telles que TOUT SELECTIONNER, LIEN RETOUR, etc.

Exemples

(1) Vous créez un formulaire qui affiche les enregistrements de la table [Annuaire]. Vous créez un objet de type onglet nommé asRolodex (avec un onglet pour chaque lettre de l'alphabet) et un sous-formulaire qui affiche les enregistrements de la table [Annuaire]. En choisissant un onglet, vous affichez les enregistrements qui correspondent à cette lettre. Puisque, dans cet exemple, la table [Annuaire] contient des données statiques, vous ne voulez pas effectuer une recherche chaque fois que vous cliquez sur un onglet et donc vous dépensez moins de temps précieux à exécuter ces recherches. Pour faire ceci, vous pouvez placer vos recherches dans les sélections temporaires pour les réutiliser quand il le faut. Vous écrivez la méthode objet de l'onglet asRolodex comme indiquée ci-dessous :

```
      ` Méthode objet de l'onglet asRolodex
Au cas ou

: (Evenement formulaire=Sur chargement)
         ` Avant que le formulaire s'affiche à l'écran,
         ` initialiser l'onglet et le tableau de booléens qui nous indiquent
         ` si une recherche pour la lettre sur laquelle vous avez cliqué
         ` a été exécutée ou pas
   TABLEAU ALPHA(1;asRolodex;26)
   TABLEAU BOOLEEN(abRechFini;26)
   Boucle ($vlElém;1;26)
      asRolodex{$vlElém}:=Caractere(64+$vlElém)
      abRechFini{$vlElém}:=Faux
   Fin de boucle
```

```
: (Evenement formulaire=Sur clic)
      ` Lorsque l'utilisateur clique sur un onglet, vérifier si une recherche pour cette
      ` lettre a été exécutée ou pas
   Si (Non(abRechFini{asRolodex}))
         ` Sinon, fixer la destination de la recherche vers une sélection temporaire
      FIXER DESTINATION RECHERCHE(Vers sélection temporaire;"Rolodex"+
                                                      asRolodex{asRolodex})
         ` Effectuer la recherche
      CHERCHER([Phone Book];[Phone Book]Last name=asRolodex{asRolodex}+"@")
         ` Restituer le mode standard de recherche
      FIXER DESTINATION RECHERCHE(Vers sélection courante)
         ` La prochaine fois que cette lettre est choisie, la recherche ne sera pas
         ` exécutée
      abRechFini{asRolodex}:=Vrai
   Fin de si
      ` Utiliser la sélection temporaire pour l'affichage des enregistrements
      `qui correspondent à cette lettre
   UTILISER SELECTION("Rolodex"+asRolodex{asRolodex})

: (Evenement formulaire=Sur libération)
      ` Après que le formulaire disparaît de l 'écran
      ` Effacer les sélections temporaires de la mémoire
   Boucle ($vlElem;1;26)
      Si(abRechFini{$vlElém})
         EFFACER SELECTION("Rolodex"+asRolodex{$vlElém})
      Fin de si
   Fin de boucle
      ` Effacer les deux tableaux dont nous n'avons pas besoin
   EFFACER VARIABLE(asRolodex)
   EFFACER VARIABLE(abRechFini)
Fin de cas
```

(2) La méthode ValeursUniques suivante vérifie si les valeurs sont uniques pour des champs dans une table de votre choix. L'enregistrement courant peut déjà exister ou vient d'être créé.

```
   ` Méthode projet ValeursUniques
   ` ValeursUniques ( Pointeur ; Pointeur { ; Pointeur... } ) -> Booléen
   ` ValeursUniques ( ->Table ; ->Champ { ; ->Champ2...  } ) -> Oui ou non

C_BOOLEEN($0;$2)
C_POINTEUR(${1})
C_ENTIER LONG($vlChamp;$vlNmbChamps;$vlTrouvé;$vlEnregCour)
$vlNmbChamps:=Nombre de parametres-1
$vlEnregCour:=Numero enregistrement($1->)
```

```
Si ($vlNmbChamps>0)
   Si ($vlEnregCour#-1)
      Si ($vlEnregCour<0)
            ` Il s'agit d'un nouvel enregistrement qui n'a pas été sauvegardé
            ` (numéro d'enregistrement est égal à -3) donc nous pouvons
            ` arrêter la recherche dès que nous avons trouvé un enregistrement
         FIXER LIMITE RECHERCHE(1)
      Sinon
            ` Il s'agit d'un enregistrement existant, donc nous pouvons arrêter
            ` la recherche dès que nous avons trouvé au moins deux enregistrements
         FIXER LIMITE RECHERCHE(2)
      Fin de si
         ` La recherche retournera le résultat dans la variable $vlTrouvé
         ` sans changer l'enregistrement courant ni la sélection courante
      FIXER DESTINATION RECHERCHE(Vers variable;$vlTrouvé)
         ` Construire la recherche selon le nombre de champs spécifiés
      Au cas ou
         : ($vlNmbChamps=1)
            CHERCHER($1->;$2->=$2->)
         : ($vlNmbChamps=2)
            CHERCHER($1->;$2->=$2->;*)
            CHERCHER($1->; & ;$3->=$3->)
         Sinon
            CHERCHER($1->;$2->=$2->;*)
            Boucle ($vlChamp;2;$vlNmbChamps-1)
               CHERCHER($1->; & ;${1+$vlChamp}->=${1+$vlChamp}->;*)
            Fin de boucle
            CHERCHER($1->; & ;${1+$vlNmbChamps}->=${1+$vlNmbChamps}->)
      Fin de cas
      FIXER DESTINATION RECHERCHE(0) ` Rétablir le mode standard de recherche
      FIXER LIMITE RECHERCHE(0) ` Enlever la limite sur la recherche
         ` Traiter le résultat de la recherche
      Au cas ou
         : ($vlTrouvé=0)
            $0:=Vrai ` Pas de valeurs dupliquées
         : ($vlTrouvé=1)
            Si ($vlEnregCour<0)
                  ` Trouvé un enregistrement existant avec les mêmes
                  `valeurs que le nouveau
               $0:=Faux
            Sinon
                  ` Pas de valeurs dupliquées, nous avons trouvé le
                  `même enregistrement
               $0:=Vrai
            Fin de si
```

```
            : ($vlTrouvé=2)
                $0:=Faux  ` Quoi que ce soit, les valeurs sont dupliquées
        Fin de cas
    Sinon
            ` Cela n'a aucun sens, signalez-le pendant le développement de la base
        Si (<>Débogage)
                ` ATTENTION ! Cette méthode a été appelée sans
                `enregistrement courant
            TRACE
        Fin de si
        $0:=Faux  ` Ne peut pas garantir le résultat
    Fin de si
Sinon
        ` Cela n'a aucun sens, signalez-le pendant le développement de la base
    Si (<>Débogage)
            ` ATTENTION ! Cette méthode a été appelée sans conditions de recherche
        TRACE
    Fin de si
    $0:=Faux  ` Ne peut pas garantir le résultat
Fin de si
```

Lorsque cette méthode est implémentée dans votre application, vous pouvez écrire le code suivant :

```
    ` ...
Si (ValeursUniques (->[Contacts];->[Contacts]Société;->[Contacts]Nom;
                                                    ->[Contacts]Prénom))
        ` Traitement de l'enregistrement qui a les valeurs uniques
Sinon
    ALERTE("Il existe déjà un contact avec ce nom pour cette société.")
Fin de si
    ` ...
```

Référence

CHERCHER, CHERCHER DANS SELECTION, CHERCHER PAR EXEMPLE, CHERCHER PAR FORMULE, CHERCHER PAR FORMULE DANS SELECTION, CHERCHER PAR SQL, CHERCHER PAR TABLEAU, FIXER LIMITE RECHERCHE.

FIXER LIMITE RECHERCHE (limite)

Paramètre	Type		Description
limite	Numérique	→	Nombre limite d'enregistrements ou 0 pour nombre illimité

Description

La commande FIXER LIMITE RECHERCHE vous permet d'indiquer à 4D d'arrêter toutes les recherches suivant l'appel de cette commande dans le process courant dès que le nombre d'enregistrements défini dans limite a été atteint.

Si, par exemple, limite est égal à 1, les recherches s'arrêteront dès qu'un enregistrement sera trouvé selon les conditions de la recherche.

Pour que les recherches soient de nouveau sans limite, appelez FIXER LIMITE RECHERCHE en fixant le paramètre limite à 0.

Attention : FIXER LIMITE RECHERCHE affecte toutes les recherches dans le process courant. N'oubliez pas d'associer toujours un appel à FIXER LIMITE RECHERCHE(limite) (lorsque limite>0) à un appel à FIXER LIMITE RECHERCHE(0) ultérieur pour rétablir les recherches sans limite.

FIXER LIMITE RECHERCHE modifie uniquement le comportement des commandes de recherche, c'est-à-dire :

- CHERCHER
- CHERCHER DANS SELECTION
- CHERCHER PAR EXEMPLE
- CHERCHER PAR FORMULE
- CHERCHER PAR FORMULE DANS SELECTION
- CHERCHER PAR SQL
- CHERCHER PAR TABLEAU

En revanche, FIXER LIMITE RECHERCHE n'affecte pas les autres commandes qui modifient la sélection courante d'une table telles que TOUT SELECTIONNER, LIEN RETOUR, etc.

Exemples

(1) Pour effectuer une recherche qui correspond à la formule "...trouver dix clients avec lesquels les ventes sont supérieures à 1MF...", écrivez le code suivant :

```
FIXER LIMITE RECHERCHE(10)
CHERCHER([Clients];[Clients]Ventes>1000000)
FIXER LIMITE RECHERCHE(0)
```

(2) Référez-vous au deuxième exemple de la commande FIXER DESTINATION RECHERCHE.

Référence

CHERCHER, CHERCHER DANS SELECTION, CHERCHER PAR EXEMPLE, CHERCHER PAR FORMULE, CHERCHER PAR FORMULE DANS SELECTION, CHERCHER PAR SQL, CHERCHER PAR TABLEAU, FIXER DESTINATION RECHERCHE.

FIXER RECHERCHE ET VERROUILLAGE (verrou)

Paramètre	Type		Description
verrou	Booléen	→	Vrai = verrouiller les enregistrements trouvés par les recherches, Faux = ne pas les verrouiller

Description

La commande FIXER RECHERCHE ET VERROUILLAGE vous permet de demander le verrouillage automatique des enregistrements trouvés par toutes les recherches qui suivent son appel dans la transaction courante. Ce mécanisme permet de s'assurer que les enregistrements ne puissent pas être modifiés par un process autre que le process courant entre une recherche et la manipulation des résultats.

Par défaut, les enregistrements trouvés par les recherches ne sont pas verrouillés. Passez Vrai dans le paramètre verrou pour activer le verrouillage.

Cette commande doit impérativement être utilisée à l'intérieur d'une transaction. Si elle est appelée hors du contexte d'une transaction, une erreur est générée. Ce principe permet un meilleur contrôle du verrouillage des enregistrements. Les enregistrements trouvés restent verrouillés tant que la transaction n'a pas été terminée (qu'elle ait été validée ou annulée). A l'issue de la transaction, tous les enregistrements sont déverrouillés.

Le verrouillage des enregistrements est effectif pour toutes les tables dans la transaction courante.

Lorsqu'une instruction FIXER RECHERCHE ET VERROUILLAGE(Vrai) a été exécutée, les commandes de recherche (par exemple CHERCHER) adoptent un fonctionnement spécifique si un enregistrement déjà verrouillé est trouvé :
- la recherche est stoppée et la variable système *OK* prend la valeur 0,
- la sélection courante est vidée,
- l'ensemble système *LockedSet* contient l'enregistrement verrouillé à l'origine de l'arrêt de la recherche.
Par conséquent, dans ce contexte il est nécessaire de tester l'ensemble *LockedSet* à l'issue d'une recherche infructueuse (sélection courante vide et/ou variable OK à 0) afin de déterminer la cause de l'échec.

Appelez FIXER RECHERCHE ET VERROUILLAGE(Faux) afin de désactiver le mécanisme après usage.

Exemple

Dans cet exemple, il n'est pas possible de supprimer un client qui aurait été passé de la catégorie "C" à la catégorie "A" par un autre process entre le CHERCHER et le SUPPRIMER SELECTION :

```
DEBUT TRANSACTION
FIXER RECHERCHE ET VERROUILLAGE(Vrai)
CHERCHER([Clients];[Clients]Catégorie="C")
   `A cet instant, les enregistrements trouvés sont automatiquement verrouillés pour
   `tous les autres process
SUPPRIMER SELECTION([Clients])
FIXER RECHERCHE ET VERROUILLAGE(Faux)
VALIDER TRANSACTION
```

Gestion des erreurs

Si la commande est appelée hors du contexte d'une transaction, une erreur est générée.

Référence

CHERCHER.

Lire dernier chemin recherche

Lire dernier chemin recherche → Chaîne

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Chaîne	←	Description du chemin de la dernière recherche exécutée

Description

La commande Lire dernier chemin recherche retourne la description interne détaillée du chemin réel de la dernière recherche effectuée sur les données. Pour plus d'informations sur les descriptions de recherches, reportez-vous à la documentation de la commande DECRIRE EXECUTION RECHERCHE.

Cette description est retournée au format Texte ou XML en fonction de la valeur passée dans le paramètre formatDesc. Vous pouvez passer une des constantes suivantes, placées dans le thème "Recherches" :

Constante	Type	Valeur
Description format Texte	Entier long	0
Description format XML	Entier long	1

Cette commande retourne une valeur significative si la commande DECRIRE EXECUTION RECHERCHE a été exécutée au cours de la session.

La description du chemin de la dernière recherche peut être comparée à la description du plan prévu de la dernière recherche (obtenue à l'aide de la commande Lire dernier plan recherche) à des fins d'optimisations.

Référence

DECRIRE EXECUTION RECHERCHE, Lire dernier plan recherche.

Lire dernier plan recherche

Lire dernier plan recherche (formatDesc) → Chaîne

Paramètre	Type		Description
formatDesc	Entier long	→	Format de description (Texte ou XML)
Résultat	Chaîne	←	Description du plan de la dernière recherche exécutée

Description

La commande Lire dernier plan recherche retourne la description interne du plan d'exécution prévu pour la dernière recherche effectuée sur les données. Pour plus d'informations sur les descriptions de recherches, reportez-vous à la commande DECRIRE EXECUTION RECHERCHE.

Cette description est retournée au format Texte ou XML en fonction de la valeur passée dans le paramètre formatDesc. Vous pouvez passer une des constantes suivantes, placées dans le thème "Recherches" :

Constante	Type	Valeur
Description format Texte	Entier long	0
Description format XML	Entier long	1

Cette commande retourne une valeur significative si la commande DECRIRE EXECUTION RECHERCHE a été exécutée au cours de la session.
La description du plan de la dernière recherche peut être comparée à la description du chemin réel de la dernière recherche (obtenue à l'aide de la commande Lire dernier chemin recherche) à des fins d'optimisations.

Référence

DECRIRE EXECUTION RECHERCHE, Lire dernier chemin recherche.

Trouver dans champ

version 11 (Modifiée)

Trouver dans champ (champCible; valeur) → Entier long

Paramètre	Type		Description
champCible	Champ	→	Champ sur lequel effectuer la recherche
valeur	Champ | Variable	→	Valeur à rechercher
		←	Valeur trouvée
Résultat	Entier long	←	Numéro de l'enregistrement trouvé ou -1 si pas d'enregistrement trouvé

Description

La commande Trouver dans champ retourne le numéro du premier enregistrement dont le champ champCible est égal à la valeur valeur.
Si aucun enregistrement ne correspond au critère, Trouver dans champ retourne -1.

Après l'appel, le paramètre valeur contient la valeur effectivement trouvée. Ce fonctionnement permet d'effectuer des recherches utilisant le caractère "@" sur des champs de type alpha, et pour lesquelles il est nécessaire de récupérer la valeur trouvée.
La commande ne modifie ni la sélection courante, ni l'enregistrement courant.

Cette fonction, très rapide, est particulièrement utile pour prévenir la création de doublons au moment de la saisie de données.

Exemple

Dans une base de données de CD audio, vous souhaitez vérifier, au moment de la saisie d'un nouveau nom de chanteur, si celui-ci n'existe pas déjà dans la base. Comme il peut exister des homonymes, vous ne souhaitez pas toutefois que le champ [Chanteur]Nom soit unique. Pour cela, dans le formulaire d'entrée, vous écrivez dans la méthode objet du champ [Chanteur]Nom :

```
Si (Evenement formulaire=Sur données modifiées)
    $EnrgNum:=Trouver dans champ([Chanteur]Nom;[Chanteur]Nom)
    Si ($EnrgNum # -1)  ` Si ce nom a déjà été saisi
        CONFIRMER("Un chanteur de ce nom existe déjà. Voulez-vous visualiser sa
                                                                fiche ?";"Oui";"Non")
        Si (OK=1)
            ALLER A ENREGISTREMENT([Chanteur];$EnrgNum)
        Fin de si
    Fin de si
Fin de si
```

TRIER ({table}{; champ}{; > ou <}{; champ2; > ou <2; ...; champN; > ou <N}{; *})

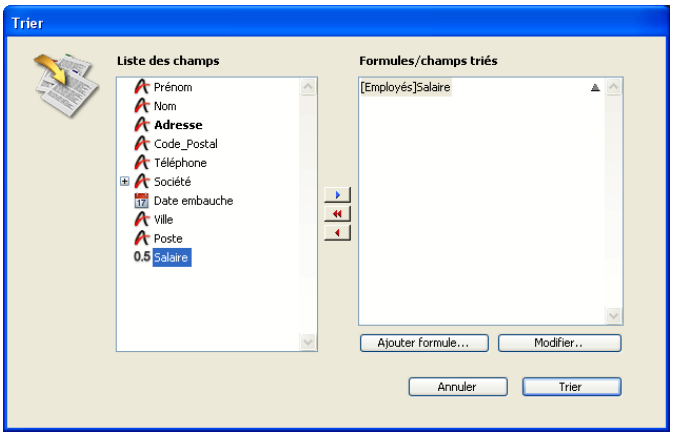
Paramètre	Type		Description
table	Table	→	Table de laquelle réordonner la sélection courante ou Table par défaut si ce paramètre est omis
champ	Champ	→	Champ sur lequel effectuer le tri pour chaque niveau
> ou <		→	Sens du tri pour chaque niveau : > demander un tri croissant ou < demander un tri décroissant
*		→	Attente d'exécution du tri

Description

TRIER trie (réordonne) les enregistrements de la sélection courante de table pour le process courant. Une fois le tri effectué, le premier enregistrement de la nouvelle sélection courante devient le nouvel enregistrement courant.

Si vous omettez le paramètre table, la commande s'applique à la table par défaut. Si aucune table par défaut n'a été définie, une erreur est retournée.

Si vous ne passez ni le paramètre champ, ni les paramètres >, < ou *, TRIER affiche la boîte de dialogue de l'Editeur de tri de 4D pour table. Cet éditeur est présenté ci-dessous :



Pour plus d'informations sur l'utilisation de cet éditeur, reportez-vous au manuel *Mode Développement* de 4D.
L'utilisateur construit le tri puis clique sur le bouton **Trier**. Si le tri est correctement effectué, la variable système OK prend la valeur 1. Si l'utilisateur clique sur **Annuler**, aucun tri n'est effectué et la variable OK prend la valeur 0 (zéro).

Exemples

(1) L'exemple suivant affiche la boîte de dialogue de Tri pour la table [Produits] :

```
TRIER ([Produits])
```

(2) L'exemple suivant affiche la boîte de dialogue de Tri pour la table par défaut (si elle a été définie) :

```
TRIER
```

Si vous spécifiez les paramètres champ et > ou <, la boîte de dialogue standard de Tri ne s'affiche pas et le tri est entièrement défini par programmation. Vous pouvez trier la sélection courante sur un plusieurs niveaux. Pour chaque niveau de tri, vous passez un champ dans le paramètre champ et un ordre de tri dans > ou <. Si vous passez le paramètre “supérieur à ” (>), l'ordre est croissant. Si vous passez le paramètre “inférieur à ” (<), l'ordre est décroissant.

Exemples

(3) L'exemple suivant trie la sélection courante de [Produits] par nom dans un ordre croissant :

```
TRIER ([Produits]; [Produits]Nom;>)
```

(4) L'exemple suivant trie la sélection courante de [Produits] par nom dans un ordre décroissant :

```
TRIER ([Produits]; [Produits]Nom;<)
```

(5) L'exemple suivant trie la sélection courante de [Produits] par type et par prix dans un ordre croissant à chaque niveau :

```
TRIER ([Produits]; [Produits]Type;>; [Produits]Prix;>)
```

(6) L'exemple suivant trie la sélection courante de [Produits] par type et par prix dans un ordre décroissant à chaque niveau :

```
TRIER ([Produits]; [Produits]Type;<; [Produits]Prix;<)
```

(7) L'exemple suivant trie la sélection courante de [Produits] par type dans un ordre croissant et par prix dans un ordre décroissant :

```
TRIER ([Produits]; [Produits]Type;>; [Produits]Prix;<)
```

(8) L'exemple suivant trie la sélection courante de [Produits] par type dans un ordre décroissant et par prix dans un ordre croissant :

```
TRIER ([Produits]; [Produits]Type;<; [Produits]Prix;>)
```

Si vous omettez le paramètre d'ordre > ou <, le tri est croissant par défaut.

Exemple

(9) L'exemple suivant trie la sélection courante de [Produits] par nom dans un ordre croissant :

```
TRIER ([Produits]; [Produits]Nom)
```

Si un seul champ est spécifié (tri sur un niveau) et s'il est indexé, le tri tire parti de l'index. Si le champ n'est pas indexé ou si plus d'un champ est utilisé, le tri est effectué de manière séquentielle (hors index composites). Le champ peut appartenir à la table de la sélection que vous triez ou à une table 1 liée à table par un lien automatique ou manuel. Dans ce cas, le tri est toujours séquentiel.
Si les champs triés sont inclus dans un index composite, TRIER tire parti de l'index.

Exemples

(10) L'exemple suivant effectue un tri indexé si le champ [Produits]Nom est indexé :

```
TRIER ([Produits]; [Produits]Nom;>)
```

(11) L'exemple suivant effectue un tri séquentiel, que les champs soient ou non indexés :

```
TRIER ([Produits]; [Produits]Type;>;[Produits]Prix;>)
```

(12) L'exemple suivant effectue un tri séquentiel à l'aide d'un champ lié :

```
TRIER ([Factures];[Société]Nom;>) ` les factures sont triées par ordre alphabétique sur le champ Nom de la société
```

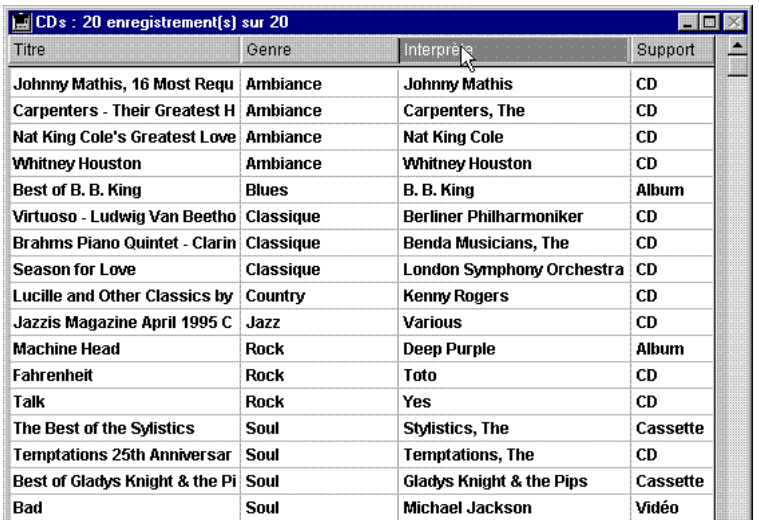
(13) L'exemple suivant effectue un tri indexé sur deux niveaux si un index composite [Contacts]Nom + [Contacts]Prénom a été défini dans la base :

```
TRIER ([Contacts];[Contacts]Nom ;> ;[Contacts]Prénom ;>)
```

Pour indiquer que le tri ne doit pas être immédiatement effectué, passez en dernier paramètre le symbole *. 4D attendra de rencontrer une nouvelle ligne de tri ne se terminant pas par * pour exécuter le tri. Cette possibilité est utile pour gérer les tris multicritères dans le cadre d'interfaces personnalisées.
Attention : lorsque vous utilisez cette syntaxe, vous ne pouvez passer qu'un seul niveau de tri (un seul champ) par ligne d'instruction.

Exemple

(14) Dans un formulaire sortie affiché en mode Application, vous souhaitez permettre aux utilisateurs de trier une colonne par ordre croissant en cliquant simplement sur son en-tête. Si l'utilisateur maintient la touche **Maj** enfoncée et clique ensuite sur plusieurs autres colonnes, le tri est multicritères, c'est-à-dire que les colonnes sont triées sur autant de niveaux qu'il y a de clics :



Titre	Genre	Interprète	Support
Johnny Mathis, 16 Most Requ	Ambiance	Johnny Mathis	CD
Carpenters - Their Greatest H	Ambiance	Carpenters, The	CD
Nat King Cole's Greatest Love	Ambiance	Nat King Cole	CD
Whitney Houston	Ambiance	Whitney Houston	CD
Best of B. B. King	Blues	B. B. King	Album
Virtuoso - Ludwig Van Beetho	Classique	Berliner Philharmoniker	CD
Brahms Piano Quintet - Clarin	Classique	Benda Musicians, The	CD
Season for Love	Classique	London Symphony Orchestra	CD
Lucille and Other Classics by	Country	Kenny Rogers	CD
Jazzis Magazine April 1995 C	Jazz	Various	CD
Machine Head	Rock	Deep Purple	Album
Fahrenheit	Rock	Toto	CD
Talk	Rock	Yes	CD
The Best of the Sylistics	Soul	Stylistics, The	Cassette
Temptations 25th Anniversar	Soul	Temptations, The	CD
Best of Gladys Knight & the Pi	Soul	Gladys Knight & the Pips	Cassette
Bad	Soul	Michael Jackson	Vidéo

Chaque en-tête de colonne contient un bouton inversé dont la méthode est du type suivant :

```
MULTITRIS (->[CDs]Titre)  `Bouton de l'en-tête de la colonne Titre
```

Chaque bouton appelle la méthode projet MULTITRIS en passant un pointeur sur le champ de la colonne. Voici le contenu de la méthode projet MULTITRIS :

```
` Méthode projet MULTITRIS
` MULTITRIS (Pointeur)
` MULTITRIS (->[Table]Champ)

C_POINTEUR($1)
C_ENTIER LONG($nbCrit)
   `Construction des critères
Si (Non(Majuscule enfoncee))  `Si le tri est simple
   TABLEAU POINTEUR(tPtrTriChp;1)  `Créons un tableau à 1 élément
   tPtrTriChp{1}:=$1  `Champ sur lequel l'utilisateur a cliqué
Sinon  `Si la touche Maj était enfoncée (tri multicritère)
   $nbCrit:=Chercher dans tableau(tPtrTriChp;$1)  `Vérifions la présence du critère
   Si ($nbCrit<0)  `Critère inexistant
         `Remplissons le tableau
      INSERER DANS TABLEAU(tPtrTriChp;Taille tableau(tPtrTriChp)+1;1)
      tPtrTriChp{Taille tableau(tPtrTriChp)}:=$1
   Fin de si
Fin de si
   `Exécution du tri
$nbCrit:=Taille tableau(tPtrTriChp)
Si ($nbCrit>0)  `S'il y a au moins un élément dans le tableau de pointeurs
   Boucle ($i;1;$nbCrit)  `Pour chaque critère défini
      TRIER([CDs];(tPtrTriChp{$i})->;>;*)  `On construit le tri
   Fin de boucle
   TRIER([CDs])  `Pas de * : on effectue le tri
Fin de si
```

Quelle que soit la manière dont le tri est défini, si l'opération risque de prendre un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Vous pouvez décider d'afficher ou de ne pas afficher ce message pour le process à l'aide des commandes SUPPRIMER MESSAGES et LAISSER MESSAGES. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton **Stop** pour interrompre l'opération. Si le tri s'est correctement déroulé, la variable système OK prend la valeur 1. Sinon, si le tri est interrompu, OK prend la valeur 0 (zéro).

Référence

TRIER PAR FORMULE.

TRIER PAR FORMULE (table{; expression{; > ou <}}{; expression2; > ou <2; ...; expressionN; > ou <N})

Paramètre	Type		Description
table	Table	→	Table de laquelle trier la sélection d'enregistrements
expression		→	Formule de tri des enregistrements (peut être de type Alphanumérique, Réel, Entier, Entier long, Date, Heure ou Booléen)
> ou <		→	Ordre de tri pour chaque niveau : > ordre croissant ou < ordre décroissant

Description

TRIER PAR FORMULE trie (réordonne) les enregistrements de la sélection courante de table pour le process courant. Une fois le tri effectué, le premier enregistrement de la nouvelle sélection courante devient le nouvel enregistrement courant.

Notez que vous devez spécifier table. Vous ne pouvez pas utiliser une table par défaut.

Vous pouvez trier la sélection sur un ou plusieurs niveaux. Pour chaque niveau, vous passez une expression dans expression et un ordre de tri dans > ou <. Si vous passez le symbole “supérieur à” (>), l'ordre est croissant. Si vous passez le symbole “inférieur à” (<), l'ordre est décroissant. Si vous ne passez pas ce paramètre, l'ordre est par défaut croissant.

Le paramètre expression peut être de type Alpha, Réel (Numérique), Entier, Entier long, Date, Heure ou Booléen.

Quelle que soit la manière dont le tri est défini, si l'opération risque de prendre un certain temps, 4D affiche automatiquement un message contenant un thermomètre de progression. Vous pouvez décider d'afficher ou de ne pas afficher ce message pour le process à l'aide des commandes SUPPRIMER MESSAGES et LAISSER MESSAGES. Si le thermomètre de progression est affiché, l'utilisateur peut cliquer sur le bouton **Stop** pour interrompre l'opération. Si le tri s'est correctement déroulé, la variable système OK prend la valeur 1. Sinon, si le tri est interrompu, OK prend la valeur 0 (zéro).

4D Server : A compter de la version 11 de 4D Server, cette commande est exécutée sur le serveur, ce qui optimise son exécution. A noter qu'en cas d'appel direct de variables dans l'expression, le tri est calculé avec la valeur de la variable sur le poste client.
En revanche, ce principe n'est pas appliqué pour les formules utilisant des méthodes qui, elles-mêmes, font appel à des variables (la valeur des variables est évaluée sur le serveur). Dans ce contexte, il peut être judicieux d'utiliser l'attribut de méthode "Exécuter sur serveur" permettant d'exécuter une méthode sur le serveur en lui passant des paramètres (variables) (cf. manuel *Mode Développement*).
Dans les versions précédentes de 4D Server, cette commande était exécutée sur les postes clients. Par compatibilité, ce fonctionnement est maintenu dans les bases de données converties en version 11. Une préférence de compatibilité et un sélecteur de la commande FIXER PARAMETRE BASE permettent toutefois d'adopter le fonctionnement de la version 11 (exécution sur le serveur) dans ces bases de données.

Exemple

L'exemple suivant trie les enregistrements de la table [Personnes] dans l'ordre décroissant par rapport à la longueur du nom de famille de chaque personne. L'enregistrement de la personne qui a le nom le plus long sera le premier enregistrement de la sélection courante :

```
TRIER PAR FORMULE ([Personnes]; Longueur ([Personnes]Nom); <)
```

Référence

TRIER.

49

Ressources

Notes de compatibilité sur les mécanismes de gestion des ressources (4D v11)

La gestion des ressources a été modifiée dans 4D à compter de la v11. Conformément aux orientations définies par Apple et mises en oeuvres dans les versions de Mac OS les plus récentes, le concept de ressources au sens strict (cf. définition ci-dessous) est désormais obsolète et va être progressivement abandonné. De nouveaux mécanismes sont mis en place pour prendre en charge les besoins précédemment couverts par les ressources : fichiers XLIFF pour la traduction des chaînes de caractères, fichiers images .png... De fait, les fichiers de ressources disparaissent au profit de fichiers de type standard. 4D accompagne cette évolution et, à partir de la v11, propose de nouveaux outils pour la gestion des traductions des bases de données, tout en maintenant la compatibilité avec les systèmes existants.

Compatibilité
Pour le maintien de la compatibilité et afin de permettre l'adaptation progressive des applications existantes, les mécanismes de gestion des ressources continuent de fonctionner dans 4D v11, à quelques différences près :
• Lorsqu'ils sont présents, les fichiers de ressources sont toujours pris en charge par 4D et le principe de la "chaîne des fichiers de ressources" (ouverture successive de plusieurs fichiers de ressources) reste valide. La "chaîne des fichiers de ressources" comprend notamment les fichiers .rsr ou .4dr. des bases de données converties (ouverts automatiquement) et les fichiers de ressources personnalisés ouverts via les commandes de ce thème.
• Toutefois, pour cause d'évolution de l'architecture interne, il n'est plus possible d'accéder directement via ces commandes (ou via les références dynamiques) aux ressources de l'application 4D ni à celles du système. Certains développeurs ont pu tirer parti de ressources internes de 4D pour leurs interfaces (par exemple les ressources contenant les noms des mois ou ceux des commandes du langage). Cette pratique est désormais à proscrire. Dans la plupart des cas, il est possible d'utiliser d'autres moyens que les ressources internes de 4D (constantes, commandes du langage...). Afin de limiter l'impact de cette modification sur les bases existantes, un système de substitution a été mis en place, basé sur l'externalisation des ressources les plus utilisées. Il est cependant fortement conseillé de faire évoluer les bases de données converties et de supprimer les appels aux ressources internes de 4D.
• Les bases de données créées avec 4D à partir de la v11 ne comportent plus par défaut de fichiers .RSR (ressources de la structure) et .4DR (ressources des données).

Nouveaux principes de gestion des ressources

Dans 4D v11, la notion de “ressources” doit désormais être entendue au sens large comme “fichiers nécessaires à la traduction de l’interface des applications”. La nouvelle architecture des ressources s’appuie sur un dossier, nommé **Resources**, impérativement situé à côté du fichier de structure de la base (.4db ou .4dc). Il n’est pas créé par défaut, vous devez le créer si votre base utilise des ressources. Vous devez placer dans ce dossier tous les fichiers nécessaires à la traduction ou la personnalisation de l’interface de l’application (fichiers image, texte, XLIFF...).

Il peut également contenir les éventuels fichiers de resources “ancienne génération” de la base (fichiers .rsr). Attention, ces fichiers ne sont pas automatiquement inclus dans la chaîne des ressources, ils devront être ouverts via les commandes standard de manipulation des ressources de 4D. 4D utilise des mécanismes automatiques pour l’exploitation du contenu de ce dossier, notamment pour la gestion des fichiers XLIFF (ce point est traité dans le manuel *Mode Développement*). Deux commandes du thème "Ressources" permettent de tirer parti de cette architecture (cf. commandes Lire chaine dans liste et LISTE DE CHAINES VERS TABLEAU).

Gestion des ressources (principes traditionnels)

Note de compatibilité : les principes traditionnels de gestion des ressources sont progressivement abandonnés dans 4D (cf. paragraphe précédent). Dans le cadre de nouvelles bases de données, il est désormais conseillé de s'appuyer sur l'architecture XLIFF ou sur l'emploi de fichiers standard.

Qu'est-ce qu'une ressource ?

Une **ressource** regroupe des données de tout type, structurées dans un format défini, et stockées dans un fichier séparé ou dans la **resource fork** ("partie de ressources") d'un fichier Mac OS. Généralement, les ressources contiennent des chaînes de caractères, des images, des icônes, etc. En fait, vous pouvez créer et utiliser vos propres types de ressources et y stocker toutes les données que vous voulez.

Data fork, Resource fork et fichier de ressources

A l'origine, sur Macintosh, les données et les ressources étaient stockées dans le même fichier, constitué d'une **data fork** ("partie de données") et d'une **resource fork** ("partie de ressources"). La data fork d'un fichier Macintosh est l'équivalent d'un fichier Windows ou UNIX. La resource fork d'un fichier Macintosh contient les ressources spécifiques Mac OS du fichier et n'a pas d'équivalent direct sous Windows ou UNIX.

Bien que ce fonctionnement soit toujours pris en charge par 4D, désormais sous Mac OS comme sous Windows, les ressources sont stockées dans un fichier séparé (de type "data fork" sous Mac OS). Ce principe est géré de façon transparente par 4D et permet d'échanger les fichiers entre les différentes plates-formes sans qu'aucune conversion ne soit nécessaire. Les commandes de gestion des fichiers de ressources (Creer fichier ressources et Ouvrir fichier ressources) permettent de travailler directement en data fork pour une meilleure compatibilité multi-plate-forme.

Fichiers de ressources
Dans les bases de données créées avec une version de 4D antérieure à la v11, 4D a créé automatiquement un fichier suffixé .rsr afin de stocker les ressources du fichier de structure et un fichier .4dr pour les ressources du fichier de données.
L'application 4D elle-même fait appel à des ressources, stockées dans un fichier suffixé ".RSR". Les plug-ins 4D, comme par exemple 4D Write, peuvent également utiliser des ressources.

En plus des fichiers de ressources fournis par 4D, vous pouvez créer et utiliser vos propres fichiers de ressources à l'aide des commandes 4D Creer fichier ressources et Ouvrir fichier ressources. Lorsque leur exécution s'est déroulée correctement, ces deux commandes retournent un **numéro de référence de fichier de ressources** identifiant de manière unique le fichier de ressources ouvert. Ce numéro équivaut au numéro de référence de document retourné, pour les fichiers standard, par les commandes du thème Documents système, telles que Ouvrir document. Toutes les commandes 4D de gestion des ressources acceptent un numéro de référence de fichier de ressources (facultatif). Une fois que vous en avez terminé avec un fichier de ressources, n'oubliez pas de le refermer en appelant la commande FERMER FICHIER RESSOURCES.

La chaîne des fichiers de ressources
Lorsque vous travaillez avec une base 4D, vous pouvez soit utiliser **tous les fichiers de ressources ouverts** soit **un fichier de ressources particulier**.

Plusieurs fichiers de ressources peuvent être ouverts simultanément. C'est d'ailleurs toujours le cas lorsqu'une base 4D est en cours d'utilisation :
• Sur Macintosh, le fichier de ressources du système est ouvert.
• Sous Windows, le fichier ASIPORT.RSR est ouvert (il contient une partie des ressources système du Macintosh).
• Le fichier de ressources de l'application 4D est ouvert.
• Le fichier de ressources de la structure de la base est ouvert (s'il existe).
• Le fichier de ressources des données de la base est ouvert (s'il existe).
• Enfin, vous pouvez ouvrir votre propre fichier de ressources à l'aide de la fonction Ouvrir fichier ressources.

Cette liste de ressources ouvertes s'appelle la **chaîne des fichiers de ressources**. Lorsque vous recherchez une ressource particulière, celle-ci peut être désignée de deux manières :
• Si vous passez un numéro de référence de fichier de ressources à une commande 4D de gestion des ressources, la ressource est recherchée dans ce fichier uniquement.
• Si vous ne passez pas de numéro de référence de fichier de ressources à la commande 4D, la ressource est recherchée dans tous les fichiers de ressources ouverts, depuis le plus récemment ouvert jusqu'au premier ouvert. 4D remonte en sens inverse la chaîne des fichiers de ressources ouverts : le dernier fichier ouvert est examiné en premier.

Types de ressources
Le format interne d'un fichier de ressources est très structuré. En plus des données de chaque ressource, le fichier contient un en-tête et un descriptif qui fournissent des informations précises sur son contenu.

Les ressources sont classées en **types**. Le type d'une ressource est indiqué par une chaîne de 4 caractères. Par exemple :
• Une ressource de type "STR#" est une ressource contenant une liste de chaînes Pascal. Elle est appelée **ressource liste de chaînes**.
• Une ressource de type "STR " (notez que le quatrième caractère est un caractère d'espacement) est une ressource contenant une chaîne Pascal individuelle. Elle est appelée **ressource chaîne**.
• Une ressource de type "TEXT" est une ressource contenant du texte sans longueur déclarée. Elle est appelée **ressource texte**.
• Une ressource de type "PICT" est une ressource contenant une image QuickDraw Macintosh que vous pouvez utiliser et afficher sous Mac OS et Windows avec 4D. Elle est appelée **ressource image**.
• Une ressource de type "cicn" est une ressource contenant une icône couleur Macintosh que vous pouvez utiliser et afficher sous Mac OS et Windows avec 4D. Une ressource "cicn" peut, par exemple, être associée à un élément d'une liste hiérarchique à l'aide de la commande CHANGER PROPRIETES ELEMENT. Elle est appelée **ressource icône couleur**.

En plus des types de ressources standard (la liste ci-dessus n'est pas exhaustive), vous pouvez créer vos propres types. Par exemple, vous pouvez décider de travailler avec des ressources du type "MTYP" (pour “Mon Type”).

Pour obtenir la liste des types de ressources présents parmi tous les fichiers ouverts ou dans un fichier particulier, utilisez la commande LISTE TYPES RESSOURCE. A l'inverse, pour obtenir la liste des ressources d'un certain type parmi tous les fichiers de ressources ouverts ou dans un fichier de ressources particulier, utilisez la commande LISTE RESSOURCES. Cette dernière retourne les numéros et les noms (cf. section suivante) de toutes les ressources d'un type particulier.

Un type de ressource est toujours indiqué par une chaîne de 4 caractères. Les caractères diacritiques et les majuscules/minuscules sont pris en compte. Par exemple, les types de ressources "Hi_!", "hi_!" et "HI_!" sont tous différents.

Important : Les types de ressources en caractères minuscules sont réservés pour le Système d'exploitation. Evitez de désigner vos propres types de ressources en utilisant des caractères minuscules.

ATTENTION : De nombreuses applications s'appuient sur le type des ressources pour traiter leur contenu. Par exemple, lorsqu'elles accèdent à une ressource "STR#", les applications s'attendent à trouver une liste de chaînes. Ne stockez pas de données atypiques dans des ressources de type standard, cela peut provoquer des erreurs système dans vos applications 4D ou dans d'autres applications.

ATTENTION : Comme un fichier de ressources est très structuré, vous ne devez pas y accéder par des commandes autres que celles de gestion des ressources. Notez que si vous passez un numéro de référence de fichier de ressources (sous forme d'une expression 4D de type heure, tout comme pour les numéros de référence de document) à une commande telle que ENVOYER PAQUET, rien ne vous en empêchera (aucune mise en garde ne vous est adressée). Mais il est très probable que le fichier de ressources soit endommagé par l'opération.

ATTENTION : Un fichier de ressources peut contenir jusqu'à 2 700 ressources individuelles. Prenez garde à ne pas de dépasser cette limite (aucune mise en garde ne vous est adressée, mais le fichier de ressources devient endommagé et inutilisable).

Nom et numéro de ressource

Toute ressource a un **nom de ressource**. Un nom de ressource peut contenir jusqu'à 255 caractères, tient compte des caractères diacritiques mais n'établit pas de distinction entre les majuscules et les minuscules. Les noms des ressources peuvent être utiles pour leur identification visuelle, mais généralement vous accèderez à une ressource par l'intermédiaire de son numéro. Les noms des ressources ne sont pas uniques, plusieurs ressources peuvent avoir le même nom.

Toute ressource a un **numéro d'identification** (on dit aussi numéro d'ID ou ID). Ce numéro d'ID est unique à l'intérieur d'un type et d'un fichier de ressources. Par exemple :
• Un fichier de ressource peut contenir une ressource "ABCD" ID=1 et une ressource "EFGH" ID=1.
• Deux fichiers de ressources peuvent contenir une ressource de même type et de même numéro.

Lorsque vous accédez à une ressource par l'intermédiaire d'une commande 4D, vous indiquez son type et son numéro. Si vous ne spécifiez pas le fichier de ressources dans lequel vous souhaitez la rechercher, la commande retournera l'occurrence de la ressource trouvée dans le premier fichier de ressource examiné. Rappelez-vous que les fichiers de ressources sont examinés dans l'ordre inverse de celui dans lequel ils ont été ouverts.

Les numéros de ressources sont compris entre -32 768 et 32 767.

Important : N'utilisez pas de numéros de ressources négatifs, ils sont réservés au Système d'exploitation. N'utilisez pas non plus de numéros situés entre 0 et 14 999, cet intervalle est réservé à 4D. Pour vos propres ressources, utilisez les numéros situés entre 15 000 et 32 767.

Pour obtenir les numéros et les noms de ressources d'un type particulier, utilisez la commande LISTE RESSOURCES.
Pour obtenir le nom d'une ressource individuelle, utilisez la commande Lire nom ressource.
Pour changer le nom d'une ressource individuelle, utilisez la commande ECRIRE NOM RESSOURCE.

Comme chaque commande 4D accepte de manière optionnelle un numéro de référence de fichier de ressources, vous pouvez facilement manipuler des ressources ayant le même type et le même numéro mais situées dans deux fichiers de ressources différents. L'exemple suivant copie toutes les ressources "PICT" d'un fichier de ressources dans un autre :

```
    ` Ouverture d'un fichier de ressources existant
$vhResFileA:=Ouvrir fichier ressources("")
Si (OK=1)
        ` Création d'un nouveau fichier de ressources
    $vhResFileB:=Creer fichier ressources("")
    Si (OK=1)
            ` Récupérer la liste des numéros et des noms de toutes les ressources de type
            ` "PICT" situées dans le fichier de ressources A
        LISTE RESSOURCES("PICT";$aiResID;$asResName;$vhResFileA)
            ` Pour chaque ressource :
        Boucle($vlElem;1;Taille tableau($aiResID))
            $viResID:=$aiResID{$vlElem}
                ` Charger la ressource du fichier A
            LIRE RESSOURCE ("PICT";$viResID;vxResData;$vhResFileA)
                ` Si la ressource peut être chargée
            Si (OK=1)
                    ` Ecrire la ressource dans le fichier B
                ECRIRE RESSOURCE ("PICT";$viResID;vxResData;$vhResFileB)
                    ` Si la ressource peut être écrite
                Si (OK=1)
                        ` Copie également du nom de la ressource...
                    ECRIRE NOM RESSOURCE("PICT;$viResID;$asResName{$vlElem};
                                                                $vhResFileB)
                        ` ...Ainsi que ses propriétés (cf. § ci-dessous)
                    $vlResAttr:=Lire proprietes ressource("PICT";$viResID;$vhResFileA)
                    ECRIRE PROPRIETES RESSOURCE("PICT";$viResID;$vlResAttr;
                                                                $vhResFileB)
                Sinon
                    ALERTE("La ressource PICT ID="+Chaine($viResID)+" ne peut pas
                                                                être créée.")
                Fin de si
            Sinon
                ALERTE("La ressource PICT ID="+Chaine($viResID)+" ne peut pas être
                                                                chargée.")
            Fin de si
        Fin de boucle
        FERMER FICHIER RESSOURCES($vhResFileB)
    Fin de si
    FERMER FICHIER RESSOURCES($vhResFileA)
Fin de si
```

Propriétés des ressources
En plus de son type, de son nom et de son numéro, une ressource possède des **propriétés** supplémentaires (aussi appelées attributs). Par exemple, une ressource peut être purgeable ou non. Cet attribut indique au Système d'exploitation si, une fois la ressource chargée en mémoire, il peut ou non la purger (c'est-à-dire l'effacer) en cas de besoin de mémoire supplémentaire. Comme le montre l'exemple précédent, il peut être important lors de la copie ou de la création d'une ressource de ne pas copier uniquement la ressource, mais également son nom et ses propriétés. Pour plus d'informations sur les propriétés des ressources, reportez-vous aux descriptions des commandes Lire proprietes ressource et ECRIRE PROPRIETES RESSOURCE.

Manipuler le contenu des ressources
Pour charger en mémoire tout type de ressource, utilisez la commande LIRE RESSOURCE, qui retourne le contenu de la ressource dans un BLOB. Pour créer ou réécrire une ressource sur disque, appelez la commande ECRIRE RESSOURCE, qui utilise le contenu du BLOB que vous passez en paramètre pour écrire le contenu de la ressource. Pour supprimer une ressource existante, utilisez la commande SUPPRIMER RESSOURCE.

Pour simplifier la manipulation des ressources, 4D dispose de commandes intégrées supplémentaires dédiées à la gestion des ressources de type standard. Ces commandes vous évitent de devoir analyser des BLOBs pour pouvoir en extraire le contenu des ressources qui vous intéressent :
• LISTE DE CHAINES VERS TABLEAU remplit un tableau Alpha ou Texte avec les chaînes de caractères contenues dans une ressource liste de chaînes.
• TABLEAU VERS LISTE DE CHAINES crée ou réécrit une ressource liste de chaînes avec les éléments d'un tableau Alpha ou Texte.
• Lire chaine dans liste retourne une chaîne particulière d'une ressource liste de chaînes.
• Lire ressource chaine retourne la chaîne d'une ressource chaîne.
• ECRIRE RESSOURCE CHAINE crée ou réécrit une ressource chaîne.
• Lire ressource texte retourne le texte d'une ressource texte.
• ECRIRE RESSOURCE TEXTE crée ou réécrit une ressource texte.
• LIRE RESSOURCE IMAGE retourne l'image d'une ressource image.
• ECRIRE RESSOURCE IMAGE crée ou réécrit une ressource image.
• LIRE RESSOURCE ICONE retourne une icône couleur en tant qu'image.

Notez que ces commandes sont fournies afin de simplifier l'emploi des ressources de type standard, mais vous pouvez parfaitement utiliser LIRE RESSOURCE et ECRIRE RESSOURCE avec des BLOBs. Par exemple, la ligne de code suivante :

```
ALERTE(Lire ressource texte(20000))
```

est équivalente (en plus court) à :

```
LIRE RESSOURCE("TEXT";20000;vxData)
Si (OK=1)
    $vlOffset:=0
    ALERTE(BLOB vers texte(vxData;UTF8 Texte sans longueur;$vlOffset;
                                                    Taille BLOB(vxData)))
Fin de si
```

Les commandes 4D et les ressources

Outre les commandes de gestion des ressources décrites dans ce chapitre, plusieurs commandes 4D vous permettent de travailler avec des fichiers de ressources :

• Sur Macintosh, DOCUMENT VERS BLOB et BLOB VERS DOCUMENT peuvent charger et écrire la totalité de la resource fork d'un fichier Macintosh.

• A l'aide des commandes CHANGER PROPRIETES ELEMENT et CHANGER PROPRIETES LISTE, vous pouvez associer des ressources images ou icônes couleur aux éléments d'une liste, ou encore utiliser des ressources icônes couleur en tant qu'icônes des éléments parents d'une liste hiérarchique (déployés/contractés).

• La commande JOUER SON joue des ressources "snd " (sous Mac OS et Windows).

• La commande CHANGER POINTEUR SOURIS peut utiliser des ressources "CURS" pour modifier l'apparence du pointeur de la souris.

Référence

Commandes du thème BLOB, Erreurs du gestionnaire de ressources du système, Lire ID ressource composant.

Creer fichier ressources (resNomFichier{; typeFichier{; *}}) → DocRef

Paramètre	Type		Description
resNomFichier	Alpha	→	Nom ou chemin d'accès complet du fichier de ressources ou chaîne vide pour afficher la boîte de dialogue standard d'enregistrement de fichiers
typeFichier	Alpha	→	Type de fichier Mac OS (chaîne de 4 caractères), ou extension de fichier Windows (chaîne de 1 à 3 caractères), ou fichier de ressources ("res " / .RES) si omis
*		→	Si passé = utiliser la data fork
Résultat	DocRef	←	Numéro de référence du fichier de ressources

Description

La commande Creer fichier ressources crée et ouvre un nouveau fichier de ressources à partir du nom ou du chemin d'accès complet que vous avez passé dans resNomFichier.

Si vous passez un nom de fichier, celui-ci sera placé dans le même dossier que le fichier de structure de la base. Passez un chemin d'accès complet pour créer un fichier de ressources dans un autre dossier.

Si le fichier existe déjà et n'est pas ouvert, Creer fichier ressources le remplace par le nouveau fichier de ressources vide. Si le fichier existant est ouvert, une erreur d'E/S est retournée.

Si vous passez une chaîne vide dans resNomFichier, la boîte de dialogue standard d'enregistrement de fichiers apparaît, permettant à l'utilisateur de choisir le nom et l'emplacement du fichier de ressources à créer. Si l'utilisateur clique sur le bouton **Annuler** dans la boîte de dialogue, aucun fichier de ressources n'est créé, Creer fichier ressources retourne une valeur nulle dans DocRef et la variable système OK prend la valeur 0.

Si le fichier de ressources est correctement créé et ouvert, Creer fichier ressources retourne son numéro de référence de fichier de ressources et la variable OK prend la valeur 1. Si le fichier de ressources ne peut pas être créé, une erreur est générée.

Sous Mac OS, le type par défaut d'un fichier créé avec Creer fichier ressources est "res ". Sous Windows, l'extension de fichier par défaut est ".res". Si vous voulez créer un fichier d'un autre type :
• Sous Mac OS, passez le type du fichier dans le paramètre optionnel typeFichier.
• Sous Windows, passez dans typeFichier une extension Windows de 1 à 3 caractères ou un type de fichier Macintosh associé à l'aide de la commande ASSOCIER TYPES FICHIER.

Par défaut, si le paramètre * n'est pas passé, la commande crée et ouvre la resource fork du fichier. Si le paramètre * est passé, la commande crée et ouvre la data fork du fichier (utilisable sous Windows et Mac OS). Pour plus d'informations, reportez-vous à la section Ressources.

N'oubliez pas d'appeler finalement FERMER FICHIER RESSOURCES pour le fichier de ressources. Notez cependant que 4D referme automatiquement tous les fichiers de ressources ouverts par l'intermédiaire de Creer fichier ressources ou Ouvrir fichier ressources lorsque vous quittez l'application ou ouvrez une autre base de données.

Exemples

(1) L'exemple suivant crée et ouvre sous Windows le fichier de ressources "MesPrefs.res" dans le dossier de la base :

```
$vhResFile:=Creer fichier ressources("MesPrefs";*)
```

Sous Mac OS, l'exemple crée et ouvre le fichier “MesPrefs”.

(2) L'exemple suivant crée et ouvre sous Windows le fichier de ressources "MesPrefs.rsr" dans le dossier de la base :

```
$vhResFile:=Creer fichier ressources("MesPrefs";"rsr")
```

Sous Mac OS, l'exemple crée et ouvre le fichier “MesPrefs”.

(3) L'exemple suivant affiche la boîte de dialogue standard d'enregistrement de fichiers :

```
$vhResFile:=Creer fichier ressources("")
Si (OK=1)
    ALERTE("Vous venez de créer “"+Document+"”.")
    FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
```

Référence

APPELER SUR ERREUR, FERMER FICHIER RESSOURCES, Ouvrir fichier ressources, Ressources.

Variables et ensembles système

Si le fichier de ressources est correctement créé et ouvert, la variable système OK prend la valeur 1. Si le fichier de ressources n'a pas pu être ouvert ou si l'utilisateur a cliqué sur le bouton **Annuler** dans la boîte de dialogue standard d'enregistrement de fichiers, la variable OK prend la valeur 0 (zéro).

Si le fichier de ressources est correctement créé et ouvert par l'intermédiaire de la boîte de dialogue standard d'enregistrement de fichiers, la variable système Document contient le chemin d'accès au fichier.

Gestion des erreurs

Si le fichier de ressources n'a pas pu être créé ou ouvert à la suite d'un problème de ressource ou d'E/S, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

ECRIRE NOM RESSOURCE

ECRIRE NOM RESSOURCE (resType; resNum; resNom{; resFichier})

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de référence de ressource (ID)
resNom	Alpha	→	Nouveau nom de la ressource
resFichier	RefDoc	→	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis

Description

ECRIRE NOM RESSOURCE modifie le nom de la ressource dont le type est passé dans resType et le numéro de référence dans resNum.

Si vous ne passez pas le paramètre resFichier, la ressource est recherchée dans les fichiers de ressources ouverts. Si vous passez un numéro de référence de fichier de ressource dans le paramètre resFichier, la ressource n'est recherchée que dans ce fichier.

Si la ressource n'existe pas, ECRIRE NOM RESSOURCE ne fait rien et la variable OK prend la valeur 0 (zéro).

ATTENTION : Il est déconseillé de modifier les noms des ressources appartenant à 4D ou aux fichiers système. Cette opération peut provoquer des erreurs système.

Note : Les noms des ressources peuvent comprendre jusqu'à 255 caractères. Ils n'établissent pas de distinction entre les majuscules et les minuscules, mais les caractères diacritiques (é, è, etc.) sont respectés.

Exemple

Référez-vous à l'exemple de la commande Lire nom ressource.

Référence

ECRIRE PROPRIETES RESSOURCE.

Variables et ensembles système

La variable système OK prend la valeur 0 si la ressource n'existe pas, sinon elle prend la valeur 1.

ECRIRE PROPRIETES RESSOURCE

ECRIRE PROPRIETES RESSOURCE (resType; resNum; resAttr{; resFichier})

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de référence de ressource (ID)
resAttr	Numérique	→	Nouveaux attributs de la ressource
resFichier	RefDoc	→	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis

Description

ECRIRE PROPRIETES RESSOURCE modifie les attributs de la ressource dont le type est passé dans le paramètre resType et le numéro de référence dans resNum.

Si vous ne passez pas le paramètre resFichier, la ressource est recherchée dans les fichiers de ressources ouverts. Si vous passez un numéro de référence de fichier de ressource dans le paramètre resFichier, la ressource n'est recherchée que dans ce fichier.

Si la ressource n'existe pas, ECRIRE PROPRIETES RESSOURCE ne fait rien et la variable OK prend la valeur 0 (zéro).

La valeur numérique passée dans resAttr doit être considérée comme une valeur binaire dont chaque bit a une signification particulière. Les constantes ci-dessous sont fournies par 4D :

Constante	Type	Valeur
Masque ressource heap système	Entier long	64
Bit ressource heap système	Entier long	6
Masque ressource purgeable	Entier long	32
Bit ressource purgeable	Entier long	5
Masque ressource verrouillée	Entier long	16
Bit ressource verrouillée	Entier long	4
Masque ressource protégée	Entier long	8
Bit ressource protégée	Entier long	3
Masque ressource préchargée	Entier long	4
Bit ressource préchargée	Entier long	2
Masque ressource modifiée	Entier long	2
Bit ressource modifiée	Entier long	1

A l'aide de ces constantes, vous pouvez définir la valeur d'attributs de ressources que vous voulez. Référez-vous aux exemples proposés à la fin de cette section.

Les attributs de ressources et leurs effets sont décrits ci-dessous :

Heap système
Si cet attribut est utilisé, la ressource sera chargée dans la mémoire système au lieu de la mémoire de 4D. Vous ne devez pas utiliser cet attribut, sauf si vous êtes absolument sûr de savoir très exactement ce que vous faites.

Purgeable
Lorsque cet attribut est utilisé, la ressource chargée peut à tout moment être purgée (c'est-à-dire supprimée) de la mémoire si de la place mémoire est nécessaire pour l'allocation d'autres données. Comme de toute manière vous chargez des ressources dans des BLOBs 4D, nous vous conseillons de rendre purgeables toutes vos propres ressources afin d'optimiser l'utilisation de la mémoire. Notez cependant que si une ressource est fréquemment appelée lors d'une session de travail, il peut être intéressant de la rendre non purgeable afin de réduire les accès disque dûs au rechargement de la ressource purgée.

Verrouillée
Si cet attribut est utilisé, la ressource ne peut pas être déplacée lorsqu'elle est chargée en mémoire. Une ressource verrouillée ne peut pas être purgée même si elle est purgeable. Le verrouillage d'une ressource a l'effet indésirable de fragmenter la mémoire. Vous ne devez pas utiliser cet attribut, sauf si vous êtes absolument sûr de savoir très exactement ce que vous faites.

Protégée
Si cet attribut est utilisé, vous ne pouvez plus modifier le nom, le numéro de référence (ID) ou le contenu de la ressource. Il n'est également plus possible de supprimer cette ressource. Vous pouvez toutefois appeler la commande ECRIRE PROPRIETES RESSOURCE pour supprimer cet attribut, puis modifier ou supprimer la ressource. La plupart du temps, vous n'aurez pas à utiliser cet attribut.

Note : Cet attribut est sans effet sous Windows.

Préchargée
Si cet attribut est utilisé, la ressource est automatiquement chargée en mémoire et le fichier dans lequel elle se trouve est ouvert. Cet attribut est utile pour optimiser le chargement des ressources lors de l'ouverture d'un fichier de ressources. Cependant, la plupart du temps, vous n'aurez pas besoin de cet attribut.

Modifiée
Si cet attribut est utilisé, la ressource reçoit une marque signifiant “doit être sauvegardée sur disque” lorsque le fichier de ressources dans lequel elle se trouve est refermé. Comme la commande de 4D ECRIRE RESSOURCE gère en interne l'écriture et la ré-écriture des ressources, vous ne devez pas utiliser cet attribut, sauf si vous êtes absolument sûr de savoir très exactement ce que vous faites.

En résumé, à moins d'être certain de savoir ce que vous faites, vous n'utiliserez généralement que l'attribut purgeable, et peut-être plus rarement les attributs préchargée et protégée.

ATTENTION : Il est déconseillé de modifier les noms des ressources appartenant à 4D ou aux fichiers système. Cette opération peut provoquer des erreurs systèmes.

Exemples

(1) Référez-vous à l'exemple de la fonction Lire nom ressource.

(2) L'exemple suivant rend la ressource 'STR#' ID=17000 purgeable et laisse les autres attributs inchangés :

```
$vlAttrRes:=Lire proprietes ressource ('STR#';17000;$vhFichierRes)
ECRIRE PROPRIETES RESSOURCE('STR#';17000;$vlAttrRes ?+
                                    Bit ressource purgeable;$vhMonFichRes)
```

(3) L'exemple suivant rend la ressource 'STR#' ID=17000 préchargée et non-purgeable :

```
ECRIRE PROPRIETES RESSOURCE('STR#';17000;Masque ressource préchargée;
                                                    $vhFichierRes)
```

(4) L'exemple suivant rend la ressource 'STR#' ID=17000 préchargée, mais purgeable:

```
ECRIRE PROPRIETES RESSOURCE('STR#';17000;Masque ressource préchargée+
                                    Masque ressource purgeable;$vhFichierRes)
```

Référence

ECRIRE NOM RESSOURCE.

Variables et ensembles système

La variable système OK prend la valeur 0 si la ressource n'existe pas, sinon elle prend la valeur 1.

ECRIRE RESSOURCE (resType; resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de ressource
resDonnées	BLOB	→	Nouveau contenu de la ressource
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou Fichier de ressources courant si omis

Description

La commande ECRIRE RESSOURCE crée ou réécrit la ressource dont vous avez passé le type et le numéro dans resType et resNum avec les données passées dans le BLOB resDonnées.

Important : Vous devez passer une chaîne de 4 caractères dans resType.

Si la ressource ne peut pas être écrite, la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est ajoutée dans ce fichier. Si vous ne passez pas le paramètre resFichier, la ressource est ajoutée au premier des fichiers de la chaîne des fichiers de ressources (c'est-à-dire le dernier fichier de ressources ouvert).

Note : La taille d'une ressource peut atteindre plusieurs mega-octets.

Indépendance de plate-forme : Rappelez-vous que vous travaillez avec des ressources issues de Mac OS. Quelle que soit la plate-forme utilisée, les valeurs internes des ressources comme des entiers longs sont stockées avec l'ordre d'octets ("byte ordering") Mac OS. Sous Windows, pour les données des ressources standard (telles que les ressources listes de chaînes et les ressources images) l'ordre des octets est automatiquement inversé ("byte swapping") si nécessaire. D'un autre côté, si vous créez et utilisez vos propres structures internes de données, c'est à vous d'effectuer l'inversion d'octets des données lorsque vous les écrivez dans un BLOB (par exemple en passant Ordre octets Macintosh à une commande telle que ENTIER LONG VERS BLOB).

Exemple

Pendant une session 4D, vous conservez des préférences utilisateur dans des variables interprocess. Pour sauvegarder ces préférences d'une session sur l'autre, vous pouvez :

• utiliser les commandes ECRIRE VARIABLES et LIRE VARIABLES pour stocker et récupérer les variables dans des documents de variables sur disque.

• utiliser les commandes VARIABLE VERS BLOB, BLOB VERS DOCUMENT, DOCUMENT VERS BLOB et BLOB VERS VARIABLE pour stocker et récupérer les variables dans des documents BLOB sur disque.

• utiliser les commandes VARIABLE VERS BLOB, ECRIRE RESSOURCE, LIRE RESSOURCE et BLOB VERS VARIABLE pour stocker et récupérer les variables dans des fichiers de ressources sur disque.

L'exemple suivant utilise la troisième possibilité.

Dans la Méthode base Sur fermeture, vous écrivez :

```
    ` Méthode base Sur fermeture
Si (Tester chemin acces("DB_Prefs")#Est un document)
    $vhResFile:=Creer fichier ressources("DB_Prefs")
Sinon
    $vhResFile:=Ouvrir fichier ressources("DB_Prefs")
Fin de si
Si (OK=1)
    VARIABLE VERS BLOB(<>vbAutoRepeat;$vxPrefData)
    VARIABLE VERS BLOB(<>vlCurTable;$vxPrefData;*)
    VARIABLE VERS BLOB(<>asDfltOption;$vxPrefData;*)
        ` et ainsi de suite...
    ECRIRE RESSOURCE("PREF";26500;$vxPrefData;$vhResFile)
    FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
```

Dans la Méthode base Sur ouverture, vous écrivez :

```
    ` Méthode base Sur ouverture
C_BOOLEEN(<>vbAutoRepeat)
C_ENTIER LONG(<>vlCurTable)
$vbDone:=Faux
$vhResFile:=Ouvrir fichier ressources("DB_Prefs")
Si (OK=1)
    LIRE RESSOURCE("PREF";26500;$vxPrefData;$vhResFile)
    Si (OK=1)
        $vlOffset:=0
        BLOB VERS VARIABLE($vxPrefData;<>vbAutoRepeat;$vlOffset)
        BLOB VERS VARIABLE($vxPrefData;<>vlCurTable;$vlOffset)
        BLOB VERS VARIABLE($vxPrefData;<>asDfltOption;$vlOffset)
            ` et ainsi de suite...
        $vbDone:=Vrai
    Fin de si
    FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
Si(Non($vbDone))
    <>vbAutoRepeat:=Faux
    <>vlCurTable:=0
    TABLEAU ALPHA(127;<>asDfltOption;0)
Fin de si
```

Référence

Commandes du thème BLOB, LIRE RESSOURCE.

Variables et ensembles système

Si la ressource est écrite, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 (zéro).

ECRIRE RESSOURCE CHAINE

ECRIRE RESSOURCE CHAINE (resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource
resDonnées	Alpha	→	Nouveau contenu de la ressource STR
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

La commande ECRIRE RESSOURCE CHAINE crée ou réécrit la ressource chaîne ("STR ") dont vous avez passé le numéro dans resNum avec la chaîne de caractères que vous avez passée dans resDonnées.

Si la ressource ne peut être ajoutée, la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est ajoutée dans ce fichier. Si vous ne passez pas le paramètre resFichier, la ressource est ajoutée au premier des fichiers de la chaîne des fichiers de ressources (c'est-à-dire le dernier fichier de ressources ouvert).

Note : Une ressource chaîne peut contenir jusqu'à 255 caractères.

Référence

ECRIRE RESSOURCE TEXTE, Lire ressource chaine.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource a été écrite, sinon elle prend la valeur 0 (zéro).

ECRIRE RESSOURCE IMAGE

ECRIRE RESSOURCE IMAGE (resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource
resDonnées	Image	→	Nouveau contenu de la ressource PICT
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

La commande ECRIRE RESSOURCE IMAGE crée ou réécrit la ressource image ("PICT") dont vous avez passé le numéro dans resNum avec l'image que vous avez passée dans resDonnées.

Si la ressource ne peut être créée, la variable système OK prend la valeur 0 (zéro).

Note : Si vous passez dans resDonnées un champ ou une variable image vide (c'est-à-dire si sa taille est nulle), la commande ne fait rien et la variable OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est ajoutée dans ce fichier. Si vous ne passez pas le paramètre resFichier, la ressource est ajoutée au premier des fichiers de la chaîne des fichiers de ressources (c'est-à-dire le dernier fichier de ressources ouvert).

Note : La taille d'une ressource image peut atteindre plusieurs méga-octets, voire davantage.

Référence

LIRE RESSOURCE IMAGE.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource a été écrite, sinon elle prend la valeur 0 (zéro).

ECRIRE RESSOURCE TEXTE

ECRIRE RESSOURCE TEXTE (resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource
resDonnées	Alpha	→	Nouveau contenu de la ressource TEXT
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

La commande ECRIRE RESSOURCE TEXTE crée ou réécrit la ressource texte ("TEXT") dont vous avez passé le numéro dans resNum avec le texte ou la chaîne de caractères que vous avez passée dans resDonnées.

Si la ressource ne peut être ajoutée, la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est ajoutée dans ce fichier. Si vous ne passez pas le paramètre resFichier, la ressource est ajoutée au premier des fichiers de la chaîne des fichiers de ressources (c'est-à-dire le dernier fichier de ressources ouvert).

Note : Une ressource texte peut contenir jusqu'à 32 000 caractères.

Référence

ECRIRE RESSOURCE CHAINE, Lire ressource texte.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource a été écrite, sinon elle prend la valeur 0 (zéro).

FERMER FICHIER RESSOURCES (resFichier)

Paramètre	Type		Description
resFichier	DocRef	→	Numéro de référence de fichier de ressources

Description

La commande FERMER FICHIER RESSOURCES referme le fichier de ressources dont vous avez passé le numéro de référence dans resFichier.

Même si vous avez ouvert plusieurs fois un fichier de ressources, il vous suffit d'appeler FERMER FICHIER RESSOURCES une seule fois pour le refermer.

Si vous appliquez FERMER FICHIER RESSOURCES au fichier de ressources de l'application 4D ou de la base, la commande le détecte et ne fait rien.

Si vous passez un numéro de référence de fichier de ressources non valide, la commande ne fait rien.

N'oubliez pas d'appeler finalement FERMER FICHIER RESSOURCES pour un fichier de ressources que vous avez ouvert à l'aide des commandes Ouvrir fichier ressources ou Creer fichier ressources. Notez cependant que 4D referme automatiquement tous les fichiers de ressources ouverts lorsque vous quittez l'application ou ouvrez une autre base de données.

Exemple

L'exemple suivant crée un fichier de ressources, ajoute une ressource de type chaîne puis referme le fichier :

```
$vhDocRef:=Creer fichier ressources("Un simple fichier")
Si (OK=1)
    ECRIRE RESSOURCE CHAINE(20000;"Une simple chaîne";$vhDocRef)
    FERMER FICHIER RESSOURCES($vhDocRef)
Fin de si
```

Référence

Creer fichier ressources, Ouvrir fichier ressources.

Lire chaine dans liste

version 11 (Modifiée)

Lire chaine dans liste (resNum; strNum{; resFichier}) → Chaîne

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource ou Attribut 'id' de l'élément 'group' (XLIFF)
strNum	Numérique	→	Numéro de chaîne ou Attribut 'id' de l'élément 'trans-unit' (XLIFF)
resFichier	DocRef	→	Numéro de référence de fichier de ressources Si omis : tous les fichiers XLIFF ou les fichiers de ressources ouverts
Résultat	Chaîne	←	Valeur de la chaîne indexée

Description

La commande Lire chaine dans liste retourne :
• soit une des chaînes stockées dans la ressource liste de chaînes ("STR#") dont vous avez passé le numéro d'ID dans resNum,
• soit une chaîne stockée dans un fichier XLIFF ouvert dont vous avez passé l'attribut 'id' de l'élément 'group' dans resNum (cf. ci-dessous "Compatibilité avec l'architecture XLIFF").

Vous passez le numéro de la chaîne dans strNum. Les chaînes d'une ressource liste de chaînes sont numérotées de 1 à N. Pour récupérer toutes les chaînes (et donc leur nombre) d'une ressource liste de chaînes, utilisez la commande LISTE DE CHAINES VERS TABLEAU.

Si la ressource n'est pas trouvée, ou si la chaîne n'est pas trouvée à l'intérieur de la ressource, une chaîne vide est retournée et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : Chaque chaîne d'une ressource liste de chaînes peut contenir jusqu'à 255 caractères.

Compatibilité avec l'architecture XLIFF

La commande Lire chaine dans liste est compatible avec l'architecture XLIFF de 4D à compter de la v11 : la commande recherche dans un premier temps les valeurs correspondant à resNum et strNum dans tous les fichiers XLIFF ouverts (si le paramètre resFichier est omis). Dans ce cas, resNum désigne l'attribut **id** de l'élément **group** et strNum désigne l'attribut **id** de l'élément **trans-unit**. Si la valeur n'est pas trouvée, la commande poursuit la recherche dans les fichiers de ressources ouverts. Pour plus d'informations sur l'architecture XLIFF dans 4D, reportez-vous au manuel *Mode Développement*.

Exemple

Reportez-vous à l'exemple de la commande Mois de.

Référence

Lire ressource chaine, Lire ressource texte, LISTE DE CHAINES VERS TABLEAU.

Variables et ensembles système

OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

Note de compatibilité : Cette commande fonctionnait avec les composants d'ancienne génération, incompatibles avec 4D version 11 et suivantes. Elle est désormais sans effet et ne doit plus être utilisée.

Lire ID ressource composant (nomComp; resType; resNumOriginal) → Numérique

Paramètre	Type		Description
nomComp	Alpha (32)	→	Nom du composant référençant la ressource
resType	Alpha (4)	→	Type de ressource (4 caractères), PICT ou STR#
resNumOriginal	Numérique	→	Numéro original de la ressource, avant installation du composant
Résultat	Numérique	←	Numéro courant de la ressource

Lire nom ressource

Lire nom ressource (resType; resNum{; resFichier}) → Alpha

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de référence de ressource (ID)
resFichier	RefDoc	→	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis
Résultat	Alpha	←	Nom de la ressource

Description

Lire nom ressource retourne le nom de la ressource dont le type est passé dans resType et le numéro de référence (ID) dans resNum.

Si vous ne passez pas le paramètre resFichier, la ressource est recherchée dans tous les fichiers de ressources ouverts. Si vous passez un numéro de référence de fichier de ressource dans le paramètre resFichier, la ressource n'est recherchée que dans ce fichier.

Si la ressource n'existe pas, Lire nom ressource retourne une chaîne vide et fixe la variable OK à 0 (zéro).

Exemple

La méthode projet suivante copie une ressource ainsi que son nom et ses attributs d'un fichier de ressources vers un autre :

```
` Méthode projet COPIER RESSOURCE
` COPIER RESSOURCE ( Alpha ; Entier long ; Heure ; Heure )
` COPIER RESSOURCE ( typeRes ; IDRes ; fichierResSource ; fichierResDest )

C_ALPHA (4;$1)
C_ENTIER LONG ($2)
C_HEURE ($3;$4)
C_BLOB ($vxResData)
```

```
LIRE RESSOURCE ($1;$2;$vxData;$3)
Si (OK=1)
    ECRIRE RESSOURCE ($1;$2;$vxData;$4)
    Si (OK=1)
        ECRIRE NOM RESSOURCE ($1;$2; Lire nom ressource ($1;$2;$3);$4)
        ECRIRE PROPRIETES RESSOURCE ($1;$2; Lire proprietes ressource ($1;$2;$3);$4)
    Fin de si
Fin de si
```

Lorsque cette méthode projet est présente dans votre application, vous pouvez écrire :

```
    ` Copier la ressource 'DATA' ID = 15000 de fichier A au fichier B
COPIER RESSOURCE ("DATA";15000;$vhFichResA;$vhFichResB)
```

Référence

ECRIRE PROPRIETES RESSOURCE.

Variables et ensembles système

La variable OK prend la valeur 0 si la ressource n'existe pas, sinon elle prend la valeur 1.

Lire proprietes ressource

Lire proprietes ressource (resType; resNum{; resFichier}) → Numérique

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de référence de ressource (ID)
resFichier	RefDoc	→	Numéro de référence du fichier de ressource ou Tous les fichiers de ressources ouverts si omis
Résultat	Numérique	←	Attributs de la ressource

Description

Lire proprietes ressource retourne les attributs de la ressource dont le type est passé dans le paramètre resType et le numéro de référence dans resNum.

Si vous ne passez pas le paramètre resFichier, la ressource est recherchée dans les fichiers de ressources ouverts. Si vous passez un numéro de référence de fichier de ressource dans le paramètre resFichier, la ressource n'est recherchée que dans ce fichier.

Si la ressource n'existe pas, Lire proprietes ressource retourne 0 (zéro) et la variable OK prend également la valeur 0 (zéro).

La valeur numérique retournée par Lire proprietes ressource doit être considérée comme une valeur binaire dont chaque bit a une signification particulière. Pour une description des attributs des ressources et leurs effets, référez-vous à la commande ECRIRE PROPRIETES RESSOURCE.

Exemple

Référez-vous à l'exemple de la commande Lire nom ressource.

Référence

ECRIRE NOM RESSOURCE.

Variables et ensembles système

La variable OK prend la valeur 0 si la ressource n'existe pas, sinon elle prend la valeur 1.

LIRE RESSOURCE

LIRE RESSOURCE (resType; resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de ressource
resDonnées	BLOB	→	Champ ou variable BLOB devant recevoir les données
		←	Contenu de la ressource
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou Tous les fichiers de ressources ouverts si omis

Description

La commande LIRE RESSOURCE retourne dans le champ ou la variable BLOB resDonnées le contenu de la ressource dont le type et le numéro sont passés dans resType et resNum.

Important : Vous devez passer une chaîne de 4 caractères dans resType.

Si la ressource n'est pas trouvée, le paramètre resDonnées est laissé inchangé et la variable OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource sera recherchée dans ce fichier seulement. Si ne passez pas le paramètre resFichier, la première occurrence de la ressource trouvée en remontant la chaîne des fichiers de ressources sera retournée.

Note : La taille d'une ressource peut atteindre plusieurs méga-octets.

Indépendance de plate-forme : Rappelez-vous que vous travaillez avec des ressources issues de Mac OS. Quelle que soit la plate-forme utilisée, les valeurs internes des ressources comme des entiers longs sont stockées avec l'ordre d'octets ("byte ordering") Mac OS. Sous Windows, pour les données des ressources standard (telles que les ressources listes de chaînes et les ressources images) l'ordre des octets est automatiquement inversé ("byte swapping") si nécessaire. D'un autre côté, si vous créez et utilisez vos propres structures internes de données, c'est à vous d'effectuer l'inversion d'octets des données lorsque vous les extrayez d'un BLOB (par exemple en passant Ordre octets Macintosh à une commande telle que BLOB vers entier long).

Exemple

Reportez-vous à l'exemple de la commande ECRIRE RESSOURCE.

Référence

Commandes du thème BLOB, ECRIRE RESSOURCE, Ressources.

Variables et ensembles système

Si la ressource est trouvée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 (zéro).

Gestion des erreurs

S'il n'y a pas assez de mémoire disponible pour charger l'image, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Lire ressource chaine

Lire ressource chaine (resNum{; resFichier}) → Alpha

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis
Résultat	Alpha	←	Contenu de la ressource STR

Description

La commande Lire ressource chaine retourne la chaîne stockée dans la ressource chaîne ("STR ") dont vous avez passé le numéro d'ID dans resNum.

Si la ressource n'est pas trouvée, une chaîne vide est retournée et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : Une ressource chaîne peut contenir jusqu'à 255 caractères.

Exemple

L'exemple suivant affiche le contenu de la ressource chaîne d'ID=20911 qui doit se trouver dans au moins un des fichiers de ressources ouverts :

```
ALERTE (Lire ressource chaine(20911))
```

Référence

ECRIRE RESSOURCE CHAINE, Lire chaine dans liste, Lire ressource texte, LISTE DE CHAINES VERS TABLEAU.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LIRE RESSOURCE ICONE (resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource icône
resDonnées	Champ | Variable	→	Champ ou variable image devant recevoir l'icône
		←	Image résultante
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis

Description

La commande LIRE RESSOURCE ICONE retourne dans le champ ou la variable image resDonnées l'icône stockée dans la ressource icône couleur ("cicn") dont vous avez passé le numéro d'ID dans resNum.

Si la ressource n'est pas trouvée, le paramètre resDonnées reste inchangé et la variable système OK prend la valeur 0 (zéro).

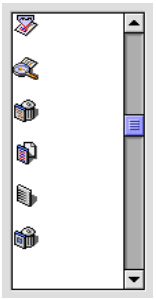
Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Exemple

L'exemple suivant charge dans un tableau image les icônes couleur situées dans l'application 4D en cours d'utilisation :

```
Si (Sous Windows)
    $vh4DResFile:=Ouvrir fichier ressources(Remplacer chaine(Fichier application;
                                                    ".EXE";".RSR"))
Sinon
    $vh4DResFile:=Ouvrir fichier ressources(Fichier application)
Fin de si
LISTE RESSOURCES("cicn";$alResID;$asResNom;$vh4DResFile)
$vlNbIcons:=Taille tableau($alResID)
TABLEAU IMAGE(ag4DIcon;$vlNbIcons)
Boucle ($vlElem;1;$vlNbIcons)
    LIRE RESSOURCE ICONE($alResID{$vlElem};ag4DIcon{$vlElem};$vh4DResFile)
Fin de boucle
```

Une fois ce code exécuté, le tableau aura l'aspect suivant lorsqu'il sera affiché dans un formulaire :



Référence

LIRE RESSOURCE IMAGE.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LIRE RESSOURCE IMAGE

LIRE RESSOURCE IMAGE (resNum; resDonnées{; resFichier})

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource
resDonnées	Champ | Variable	→	Champ ou variable image devant recevoir l'image
		←	Contenu de la ressource PICT
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis

Description

La commande LIRE RESSOURCE IMAGE retourne dans le champ ou la variable image désigné(e) par resDonnées l'image stockée dans la ressource image ("PICT") dont vous passé le numéro dans resNum.

Si la ressource n'est pas trouvée, resDonnées n'est pas modifié et la variable OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : La taille d'une ressource image peut atteindre plusieurs méga-octets, voire davantage.

Exemple

Reportez-vous à l'exemple de la commande LISTE RESSOURCES.

Référence

APPELER SUR ERREUR, ECRIRE RESSOURCE IMAGE, LIRE RESSOURCE ICONE.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

Gestion des erreurs

S'il n'y a pas assez de mémoire disponible pour charger l'image, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Lire ressource texte

Lire ressource texte (resNum{; resFichier}) → Texte

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis
Résultat	Texte	←	Contenu de la ressource TEXT

Description

La commande Lire ressource texte retourne le texte stocké dans la ressource texte ("TEXT") dont vous avez passé le numéro d'ID dans resNum.

Si la ressource n'est pas trouvée, un texte vide est retourné et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Note : Une ressource texte peut contenir jusqu'à 32 000 caractères.

Exemple

L'exemple suivant affiche le contenu de la ressource texte d'ID=20800 qui doit se trouver dans au moins un des fichiers de ressources ouverts :

```
ALERTE (Lire ressource texte(20800))
```

Référence

ECRIRE RESSOURCE TEXTE, Lire chaine dans liste, Lire ressource chaine, LISTE DE CHAINES VERS TABLEAU.

Variables et ensembles système

OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LISTE DE CHAINES VERS TABLEAU (resNum; chaînes{; resFichier})

Paramètre	Type		Description
resNum	Numérique	→	Numéro de ressource ou Attribut 'id' de l'élément 'group' (XLIFF)
chaînes	Tableau chaîne	←	Chaînes de la ressource STR# ou Chaînes de l'élément 'group' (XLIFF)
resFichier	DocRef	→	Numéro de référence de fichier de ressources Si omis : tous les fichiers XLIFF ou les fichiers de ressources ouverts

Description

La commande LISTE DE CHAINES VERS TABLEAU remplit le tableau chaînes avec :
• soit les chaînes stockées dans la ressource de type liste de chaînes ("STR#") dont vous avez passé le numéro d'ID dans resNum.
• soit une chaîne stockée dans un fichier XLIFF ouvert dont vous avez passé l'attribut 'id' de l'élément 'group' dans resNum (cf. ci-dessous "Compatibilité avec l'architecture XLIFF").

Si la ressource n'est pas trouvée, le tableau chaînes reste inchangé et la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est recherchée dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, c'est la première occurrence de la ressource rencontrée dans la chaîne des fichiers de ressources qui sera retournée.

Si vous ne pré-déclarez pas le tableau chaînes avant d'appeler LISTE DE CHAINES VERS TABLEAU, la commande crée un tableau de type Texte. Si vous pré-déclarez le tableau, vous pouvez lui assigner le type Alpha ou Texte.

Note : Chaque chaîne d'une ressource liste de chaînes peut contenir jusqu'à 255 caractères.

Conseil : Lorsque vous utilisez des ressources listes de chaînes, limitez-vous à des ressources de 32 Ko maximum et quelques centaines de chaînes par ressource.

Compatibilité avec l'architecture XLIFF

La commande LISTE DE CHAINES VERS TABLEAU est compatible avec l'architecture XLIFF de 4D v11 : la commande recherche dans un premier temps la valeur correspondant à resNum dans tous les fichiers XLIFF ouverts (si le paramètre resFichier est omis) et remplit le tableau chaînes avec les valeurs correspondantes. Dans ce cas, resNum désigne l'attribut **id** de l'élément **group** et le tableau chaînes contient toutes les chaînes de l'élément. Si la valeur n'est pas trouvée, la commande poursuit la recherche dans les fichiers de ressources ouverts. Pour plus d'informations sur l'architecture XLIFF dans 4D, reportez-vous au manuel *Mode Développement*.

Exemple

Reportez-vous à l'exemple de la commande TABLEAU VERS LISTE DE CHAINES.

Référence

Lire chaine dans liste, Lire ressource chaine, Lire ressource texte, TABLEAU VERS LISTE DE CHAINES.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource est trouvée, sinon elle prend la valeur 0 (zéro).

LISTE RESSOURCES (resType; resNums; resNoms{; resFichier})

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNums	Tableau E. long	←	Numéros des ressources de ce type
resNoms	Tableau alpha	←	Noms des ressources de ce type
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts si ce paramètre est omis

Description

La commande LISTE RESSOURCES remplit les tableaux resNums et resNoms avec les numéros et les noms des ressources dont vous avez passé le type dans resType.

Important : Vous devez passez dans resType une chaîne de 4 caractères.

Si vous passez un numéro de référence de fichier de ressources valide dans le paramètre optionnel resFichier, seules les ressources présentes dans ce fichier seront listées. Si vous ne passez pas le paramètre resFichier, toutes les ressources de tous les fichiers de ressources ouverts seront listées.

Si vous ne pré-déclarez pas les tableaux resNums et resNoms avant d'appeler LISTE RESSOURCES, la commande créera par défaut le tableau resNums avec le type Entier long et resNoms avec le type Texte. Si vous pré-déclarez les tableaux, vous devez attribuer le type Entier long à resNums, mais pouvez attribuer le type Alpha ou Texte à resNoms.

Après l'appel, vous pouvez tester le nombre de ressources qui ont été trouvées en appliquant la commande Taille tableau au tableau resNums ou resNoms.

Exemples

(1) L'exemple suivant remplit les tableaux $alResNum et $atResNom avec les numéros et les noms des ressources de type Listes de chaînes présentes dans le fichier de structure de la base :

```
Si (Sous Windows)
    $vhStructureResFile:=Ouvrir fichier ressources(Remplacer chaine
                                                   (Fichier structure;".4DB";".RSR"))
Sinon
    $vhStructureResFile:=Ouvrir fichier ressources(Fichier structure)
Fin de si
```

```
Si (OK=1)
    LISTE RESSOURCES("STR#";$alResNum;$atResNom;$vhStructureResFile)
Fin de si
```

(2) L'exemple suivant copie dans la bibliothèque d'images de la base les ressources image présentes dans tous les fichiers de ressources ouverts :

```
LISTE RESSOURCES("PICT";$alResNum;$atResNom)
Creer fenetre(50;50;550;120;5;"Copie des ressources PICT...")
Boucle ($vlElem;1;Taille tableau($alResNum))
    LIRE RESSOURCE IMAGE($alResNum{$vlElem};$vgImage)
    Si (OK=1)
        $vsNom:=$atResNom{$vlElem}
        Si ($vsNom="")
            $vsNom:="PICT resID="+Chaine($alResNum{$vlElem})
        Fin de si
        EFFACER FENETRE
        POSITION MESSAGE(2;1)
        MESSAGE("Ajout de l'image “"+$vsNom+"” à la bibliothèque d'images de la
                                                                    base.")
        ECRIRE IMAGE DANS BIBLIOTHEQUE($vgImage;$alResNum{$vlElem};$vsNom)
    Fin de si
Fin de boucle
FERMER FENETRE
```

Référence

LISTE TYPES RESSOURCE.

LISTE TYPES RESSOURCE (resTypes{; resFichier})

Paramètre	Type		Description
resTypes	Tableau alpha	←	Liste des types de ressources disponibles
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou tous les fichiers de ressources ouverts (si ce paramètre est omis)

Description

La commande LISTE TYPES RESSOURCE remplit le tableau resTypes avec les types des ressources présentes dans le(s) fichier(s) de resources ouvert(s).

Si vous passez un numéro de référence de fichier de ressources valide dans le paramètre optionnel resFichier, seules les ressources présentes dans ce fichier seront listées. Si vous ne passez pas le paramètre resFichier, toutes les ressources de tous les fichiers de ressources ouverts seront listées.

Si vous ne pré-déclarez pas le tableau resTypes avant d'appeler LISTE TYPES RESSOURCE, la commande créera par défaut un tableau de type Texte. Si vous pré-déclarez le tableau, vous pouvez lui attribuer le type Alpha ou Texte.

Après l'appel, vous pouvez tester le nombre de types de ressources différents qui ont été trouvés en appliquant la commande Taille tableau au tableau resTypes.

Exemples

(1) L'exemple suivant remplit le tableau atResType avec les types de ressources présents dans tous les fichiers de ressource ouverts :

```
LISTE TYPES RESSOURCE(atResType)
```

(2) L'exemple suivant vous indique si le fichier de structure Mac OS que vous utilisez contient des plug-ins 4D "ancien modèle", qui devront être mis à jour si vous voulez exploiter la base sous Windows :

```
$vhResFile:=Ouvrir fichier ressources(Fichier structure)
LISTE TYPES RESSOURCE(atResType;$vhResFile)
Si (Chercher dans tableau(atResType;"4DEX")>0)
   ALERTE("Cette base contient des plug-ins 4D basés sur l'ancien système."+
                        (Caractere(13)*2)+"Vous devrez les mettre à jour pour pouvoir
                                                    utiliser la base sous Windows.")
Fin de si
```

Note : Le fichier de structure n'est pas le seul fichier dans lequel des plug-ins "ancien modèle" ont pu être installés. La base peut également être associée à un fichier "Routines Externes" ou "Proc.Ext".
(3) La méthode projet suivante retourne le nombre de ressources présentes dans un fichier de ressources :

```
` Méthode projet Compter ressources
` Compter ressources ( Heure ) -> Entier long
` Compter ressources ( DocRef ) -> Nombre de ressources

C_ENTIER LONG($0)
C_HEURE($1)

$0:=0
LISTE TYPES RESSOURCE($atResType;$1)
Boucle ($vlElem;1;Taille tableau($atResType))
   LISTE RESSOURCES($atResType{$vlElem};$alResID;$atResName;$1)
   $0:=$0+Taille tableau($alResID)
Fin de boucle
```

Une fois que cette méthode est implémentée dans votre base, vous pouvez écrire par exemple :

```
$vhResFile:=Ouvrir fichier ressources("")
Si (OK=1)
   ALERTE("Le fichier “"+Document+"” contient "+
                              Chaine(Compter ressources ($vhResFile))+ressource(s).")
   FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
```

Référence

LISTE RESSOURCES.

Ouvrir fichier ressources (resNomFichier{; typeFichier}) → DocRef

Paramètre	Type		Description
resNomFichier	Alpha	→	Nom ou chemin d'accès complet du fichier de ressources ou chaîne vide pour afficher la boîte de dialogue standard d'ouverture de fichiers
typeFichier	Alpha	→	Type de fichier Mac OS (chaîne de 4 caractères), ou extension de fichier Windows (chaîne de 1 à 3 caractères), ou fichier de ressources ("res " / .RES) si omis
Résultat	DocRef	←	Numéro de référence du fichier de ressources

Description

La commande Ouvrir fichier ressources ouvre le fichier de ressources dont vous avez passé le nom ou le chemin d'accès complet dans le paramètre resNomFichier.

Si vous passez un nom de fichier, celui-ci doit se trouver dans le même dossier/répertoire que le fichier de structure de la base. Pour ouvrir un fichier de ressources se trouvant dans un autre dossier, passez un chemin d'accès complet dans resNomFichier.

Si vous passez une chaîne vide dans resNomFichier, la boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de sélectionner le fichier à ouvrir. Si l'utilisateur clique sur **Annuler** dans cette boîte de dialogue, aucun fichier de ressources n'est ouvert, Ouvrir fichier ressources retourne une valeur nulle dans RefDoc et la variable OK prend la valeur 0.

Par défaut, la commande ouvre la resource fork du fichier passé en paramètre. Si celle-ci est vide, la commande ouvre la data fork — si elle contient des ressources. Pour plus d'informations sur ce point, reportez-vous à la section Ressources.

Si le fichier de ressources est correctement ouvert, Ouvrir fichier ressources retourne son numéro de référence de fichier et met la variable OK à 1. Si le fichier de ressources n'existe pas ou si le fichier de que vous tentez d'ouvrir n'est pas un fichier de ressources, une erreur est générée.

Sous Mac OS, si vous utilisez la boîte de dialogue standard d'ouverture de fichiers, tous les fichiers sont présentés par défaut. Pour ne faire apparaître que les fichiers d'un type particulier, spécifiez-le dans le paramètre optionnel typeFichier.

Sous Windows, si vous utilisez la boîte de dialogue standard d'ouverture de fichiers, tous les fichiers sont présentés par défaut. Pour ne faire apparaître que les fichiers d'un type particulier, passez dans typeFichier une extension de fichier Windows de 1 à 3 caractères ou un type de fichier Macintosh associé à une extension Windows à l'aide de la commande ASSOCIER TYPES FICHIER.

N'oubliez pas d'appeler finalement FERMER FICHIER RESSOURCES pour le fichier de ressources. Notez cependant que 4D referme automatiquement tous les fichiers de ressources ouverts par l'intermédiaire de Ouvrir fichier ressources ou Creer fichier ressources lorsque vous quittez l'application ou ouvrez une autre base de données.

A la différence de la commande Ouvrir document qui ouvre par défaut un document avec un accès exclusif en lecture-écriture, Ouvrir fichier ressources vous permet d'ouvrir un fichier de ressources déjà ouvert dans la session 4D. Par exemple, lorsque vous tentez d'ouvrir deux fois le même document avec Ouvrir document, une erreur d'E/S vous est retournée lors de la seconde opération. En revanche, vous pouvez accéder à un fichier de ressources déjà ouvert lors de la session 4D : Ouvrir fichier ressources retourne son numéro de référence. Même lorsque vous ouvrez plusieurs fois un fichier de ressources, il vous suffit d'appeler FERMER FICHIER RESSOURCES une seule fois pour refermer ce fichier. Notez que ce fonctionnement n'est valable que lorsque le fichier de ressources est ouvert à l'intérieur de la session 4D. Si vous tentez d'ouvrir un fichier de ressources déjà ouvert par une autre application, une erreur d'E/S vous sera retournée.

ATTENTION
• Il est interdit d'accéder aux fichiers de ressources des applications 4D et des bases fusionnées avec 4D Desktop.
• Bien que techniquement possible, l'accès au fichier de ressources de la structure de la base est fortement déconseillé car ce fonctionnement devient caduc lorsque la base est compilée et fusionnée avec 4D Desktop.
Si toutefois vous accédez au fichier de ressources de la structure et souhaitez ajouter, supprimer ou modifier des ressources par programmation, pensez à tester l'environnement dans lequel la base s'exécute. Avec 4D Server, cela posera certainement d'épineux problèmes. Si, par exemple, vous modifiez une ressource sur le poste serveur (via une méthode base ou une procédure stockée), vous allez en définitive perturber le système d'administration de 4D Server chargé de distribuer de manière transparente les ressources aux postes clients. Notez qu'avec 4D Client vous n'accédez pas directement au fichier de structure : il est situé sur le poste serveur.
• Pour toutes ces raisons, si vous exploitez des ressources, vous devez les stocker dans vos propres fichiers.

• Lorsque vous travaillez avec vos propres ressources, n'utilisez pas de numéros de ressources négatifs, ils sont réservés au Système d'exploitation. N'utilisez pas non plus de numéros situés entre 0 et 14 999, cet intervalle est réservé à 4D. Pour vos propres ressources, utilisez les numéros situés entre 15 000 et 32 767. Rappelez-vous que dès qu'un fichier de ressources est ouvert, il devient le premier maillon de la chaîne des fichiers de ressources, et c'est dans ce fichier que les ressources seront recherchées en premier lieu. En conséquence, si vous stockez dans ce fichier des ressources dont les numéros appartiennent aux intervalles réservés au Système ou à 4D, ces ressources seront utilisées non seulement par les commandes telles que LIRE RESSOURCE mais également par les routines internes de l'application 4D elle-même. Si vous n'êtes pas absolument certain de ce que vous faites, n'utilisez pas les intervalles réservés, cela peut conduire à des erreurs système.
• Un fichier de ressources est très structuré et ne peut contenir plus 2 700 ressources. Si vous travaillez avec des fichiers comportant un grand nombre de ressources, il est conseillé de tester ce nombre avant d'ajouter de nouvelles ressources à un fichier (reportez-vous à l'exemple Nombre de ressources dans la description de la commande LISTE TYPES RESSOURCE).

Une fois que vous avez ouvert un fichier de ressources, vous pouvez analyser son contenu à l'aide des commandes LISTE TYPES RESSOURCE et LISTE RESSOURCES.

Exemples

(1) Dans l'exemple suivant, nous cherchons à ouvrir sous Windows le fichier de ressources "MesPrefs.res" situé dans le dossier de la base :

```
$vhResFile:=Ouvrir fichier ressources("MesPrefs";"res ")
```

Sous Mac OS, l'exemple recherchera le fichier “MesPrefs”.

(2) Cet exemple tente d'ouvrir sous Windows le fichier de ressources "MesPrefs.rsr" situé dans le dossier de la base :

```
$vhResFile:=Ouvrir fichier ressources("MesPrefs";"rsr")
```

Sous Mac OS, l'exemple recherchera le fichier “MesPrefs”.

(3) L'exemple suivant fait apparaître la boîte de dialogue standard d'ouverture de fichiers, dans laquelle tous les types de documents sont affichés :

```
$vhResFile:=Ouvrir fichier ressources("")
```

(4) L'exemple suivant fait apparaître la boîte de dialogue standard d'ouverture de fichiers, dans laquelle seuls les documents créés à l'aide de la fonction Creer fichier ressources et utilisant le type par défaut sont affichés :

```
$vhResFile:=Ouvrir fichier ressources("";"res ")
Si (OK=1)
   ALERTE("Vous venez d'ouvrir "+Document+".")
   FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
```

Référence

Creer fichier ressources, FERMER FICHIER RESSOURCES, Ressources.

Variables et ensembles système

Si le fichier de ressources est correctement ouvert, la variable système OK prend la valeur 1. Si le fichier de ressources n'a pas pu être ouvert ou si l'utilisateur a cliqué sur le bouton Annuler dans la boîte de dialogue standard d'ouverture de fichiers, la variable OK prend la valeur 0 (zéro).

Si le fichier de ressources est correctement ouvert par l'intermédiaire de la boîte de dialogue standard d'ouverture de fichiers, la variable système Document contient le chemin d'accès au fichier.

Gestion des erreurs

Si le fichier de ressources n'a pas pu être ouvert à la suite d'un problème de ressource ou d'E/S, une erreur est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

SUPPRIMER RESSOURCE (resType; resNum{; resFichier})

Paramètre	Type		Description
resType	Alpha	→	Type de ressource (4 caractères)
resNum	Numérique	→	Numéro de ressource
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

La commande SUPPRIMER RESSOURCE supprime la ressource dont vous passez le type dans le paramètre resType et le numéro dans resNum.

Si vous passez un numéro de référence de fichier de ressources dans le paramètre resFichier, 4D recherche la ressource dans ce fichier uniquement. Si vous ne passez pas le paramètre resFichier, 4D recherche la ressource dans tous les fichiers de ressources ouverts.

Si la ressource n'existe pas, SUPPRIMER RESSOURCE ne fait rien et la variable OK prend la valeur 0 (zéro). Si la ressource est correctement identifiée et effacée, la variable système OK prend la valeur 1.

ATTENTION : Ne supprimez pas de ressources appartenant à 4D ou à un fichier du système. Cette opération peut provoquer l'apparition d'erreurs système.

Exemples

(1) L'exemple suivant supprime la ressource "STR#" d'ID=20000:

```
      ` Notez que cet exemple supprimera la première ressource "STR#" ID=20000 rencontrée
      ` parmi tous les fichiers de ressources actuellement ouverts :
SUPPRIMER RESSOURCE ("STR#";20000)
```

(2) L'exemple suivant supprime la ressource "STR#" d'ID=20000 si celle-ci est présente dans un fichier particulier :

```
      ` Notez que cet exemple supprimera la ressource "STR#" d'ID=20000
      ` seulement si elle est présente dans le fichier de ressources désigné par $vhResFile :
SUPPRIMER RESSOURCE ("STR#";20000;$vhResFile)
      ` Notez également que si une ressource identique existe dans un fichier de
      ` ressources ouvert autre que le fichier spécifié par vhResFile, elle restera inchangée
```

(3) La méthode projet SUPPRIMER RESSOURCES DE TYPE supprime du fichier de ressources spécifié par le premier paramètre toutes les ressources du type spécifié par le second paramètre :

```
` Méthode projet SUPPRIMER RESSOURCES DE TYPE
` SUPPRIMER RESSOURCES DE TYPE ( Heure ; Alpha )
` SUPPRIMER RESSOURCES DE TYPE ( resFichier ; resType )

C_HEURE($1)
C_ALPHA(4;$2)

LISTE RESSOURCES($2;$aiResID;$asResNom;$1)
Si(OK=1)
   Boucle($vlElem;1;Taille tableau($aiResID))
      SUPPRIMER RESSOURCE($2;$aiResID{$vlElem};$1)
   Fin de boucle
Fin de si
```

Une fois que cette méthode projet existe dans votre base, vous pouvez écrire :

```
   ` Supprimer toutes les ressources de type "PREF" du fichier de ressources $vhResFile
SUPPRIMER RESSOURCES DE TYPE ($vhResFile;"PREF")
```

(4) La méthode projet SUPPRIMER RESSOURCE PAR NOM supprime une ressource (d'un type spécifique) dont vous connaissez le nom :

```
` Méthode projet SUPPRIMER RESSOURCE PAR NOM
` SUPPRIMER RESSOURCE PAR NOM ( Heure ; Alpha ; Alpha )
` SUPPRIMER RESSOURCE PAR NOM ( resFichier ; resType ; resNom )

C_HEURE($1)
C_ALPHA(4;$2)
C_ALPHA(255;$3)

LISTE RESSOURCES($2;$aiResID;$asResName;$1)
Si(OK=1)
   $vlElem:=Chercher dans tableau($asResName;$3)
   Si($vlElem>0)
      SUPPRIMER RESSOURCE($2;$aiResID{$vlElem};$1)
   Fin de boucle
Fin de si
```

Une fois que cette méthode projet existe dans votre base, vous pouvez écrire :

```
      ` Supprimer du fichier de ressources $vhResFile la ressource "PREF" dont le nom
      ` est “Réglages standard” :
SUPPRIMER RESSOURCE PAR NOM ($vhResFile;"PREF";"Réglages standard")
```

Référence

ECRIRE PROPRIETES RESSOURCE, LISTE RESSOURCES.

Variables et ensembles système

La variable OK prend la valeur 0 si la ressource n'existe pas ; si la ressource a été supprimée, OK prend la valeur 1.

TABLEAU VERS LISTE DE CHAINES (chaînes; resNum{; resFichier})

Paramètre	Type		Description
chaînes	Tableau alpha	→	Tableau alpha ou texte (nouveau contenu de la ressource STR#)
resNum	Numérique	→	Numéro de ressource
resFichier	DocRef	→	Numéro de référence de fichier de ressources ou fichier de ressources courant si ce paramètre est omis

Description

La commande TABLEAU VERS LISTE DE CHAINES crée ou réécrit la ressource liste de chaînes ("STR#") dont vous avez passé le numéro dans resNum. Le contenu de la ressource est créé à partir des chaînes de caractères que vous avez passées dans le tableau chaînes. Le tableau peut être de type Alpha ou Texte.

Si la ressource ne peut être ajoutée, la variable système OK prend la valeur 0 (zéro).

Si vous passez un numéro de référence de fichier de ressources valide dans resFichier, la ressource est ajoutée dans ce fichier. Si vous ne passez pas le paramètre resFichier, la ressource est ajoutée au premier des fichiers de la chaîne des fichiers de ressources (c'est-à-dire le dernier fichier de ressources ouvert).

Note : Chaque chaîne d'une ressource liste de chaînes peut contenir jusqu'à 255 caractères.

Conseil : Lorsque vous utilisez des ressources listes de chaînes, limitez-vous à des ressources de 32 Ko maximum et quelques centaines de chaînes par ressource.

Exemple

Votre base s'appuie sur un même ensemble de polices. Dans la Méthode base Sur fermeture, vous pouvez écrire :

```
    ` Méthode base Sur fermeture
Si (<>vbPolicesOK)
    LISTE DES POLICES($atFont)
    $vhResFile:=Ouvrir fichier ressources("EnsemblePolices")
    Si (OK=1)
        TABLEAU VERS LISTE DE CHAINES($atFont;15000;$vhResFile)
        FERMER FICHIER RESSOURCES($vhResFile)
    Fin de si
Fin de si
```

Dans la Méthode base Sur ouverture, vous pouvez écrire :

```
    ` Méthode base Sur ouverture
<>vbPolicesOK:=Faux
LISTE DES POLICES($atNewFont)
Si (Tester chemin acces("EnsemblePolices")#Est un document)
    $vhResFile:=Creer fichier ressources("EnsemblePolices")
Sinon
    $vhResFile:=Ouvrir fichier ressources("EnsemblePolices")
Fin de si
Si (OK=1)
    LISTE DE CHAINES VERS TABLEAU(15000;$atOldFont;$vhResFile)
    Si (OK=1)
        <>vbFontsAreOK:=Vrai
        Boucle($vlElem;1;Taille tableau($atNewFont))
            Si ($atNewFont{$vlElem}#$atOldFont{$vlElem}))
                $vlElem:=MAXLONG
                <>vbPolicesOK:=Faux
            Fin de si
        Fin de boucle
    Sinon
        <>vbPolicesOK:=Vrai
    Fin de si
    FERMER FICHIER RESSOURCES($vhResFile)
Fin de si
Si(Non(<>vbPolicesOK))
    CONFIRMER("Vous n'utilisez pas le même ensemble de polices, OK?")
    Si(OK=1)
        <>vbPolicesOK:=Vrai
    Sinon
        QUITTER 4D
    Fin de si
Fin de si
```

Référence

ECRIRE RESSOURCE CHAINE, ECRIRE RESSOURCE TEXTE, LISTE DE CHAINES VERS TABLEAU.

Variables et ensembles système

La variable système OK prend la valeur 1 si la ressource a été écrite, sinon elle prend la valeur 0 (zéro).

50

Saisie

AJOUTER ENREGISTREMENT ({table}{; }{*})

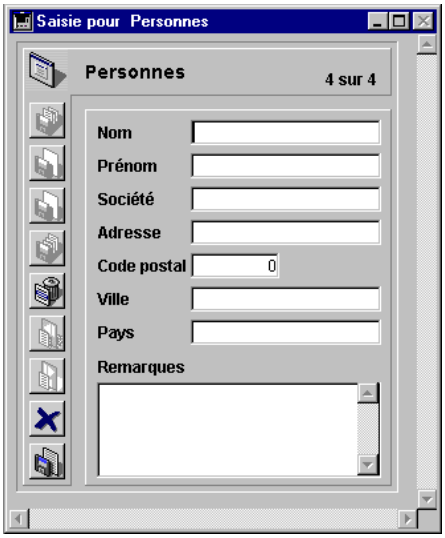
Paramètre	Type		Description
table	Table	→	Table dans laquelle ajouter des données ou Table par défaut si ce paramètre est omis
*		→	Cacher les barres de défilement

Description

La commande AJOUTER ENREGISTREMENT permet à l'utilisateur de créer un nouvel enregistrement dans table ou dans la table par défaut si ce paramètre est omis.

AJOUTER ENREGISTREMENT crée un nouvel enregistrement pour table, en fait l'enregistrement courant pour le process courant et l'affiche dans le formulaire entrée courant. En mode Application, une fois que l'utilisateur a validé le nouvel enregistrement, la sélection courante est réduite à ce seul enregistrement.

L'écran suivant présente un formulaire typiquement utilisé pour la saisie de données :



Le formulaire est affiché dans la fenêtre se trouvant au premier plan du process. Elle comporte des barres de défilement et une case de contrôle de taille. Si vous passez le paramètre optionnel astérisque (*), ces éléments n'apparaîtront pas.

AJOUTER ENREGISTREMENT affiche le formulaire jusqu'à ce que l'utilisateur valide ou annule l'enregistrement. Si l'utilisateur ajoute plusieurs enregistrements, la commande doit être appelée pour chaque nouvel enregistrement.

L'enregistrement est sauvegardé si l'utilisateur clique sur un bouton du type Valider ou appuie sur la touche Entrée, ou encore si la commande VALIDER est exécutée.

L'enregistrement n'est pas sauvegardé si l'utilisateur clique sur un bouton du type **Annuler** ou appuie sur la touche d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou encore si la commande NE PAS VALIDER est exécutée.

Après un appel à AJOUTER ENREGISTREMENT, la variable système OK prend la valeur 1 si l'enregistrement est validé et 0 s'il est annulé.

Note : Même lorsqu'il est annulé, l'enregistrement reste en mémoire et peut être sauvegardé avec la commande STOCKER ENREGISTREMENT si celle-ci est exécutée avant que le pointeur d'enregistrement courant ne soit modifié.

Exemples

(1) L'exemple suivant est une boucle souvent utilisée pour créer des enregistrements dans une base :

```
FORMULAIRE ENTREE ([Clients]; "SaisieClients")
   ` Désigner le formulaire entrée de la table [Clients]
Repeter ` Boucle jusqu'à ce que l'utilisateur annule
   AJOUTER ENREGISTREMENT ([Clients];*)
      ` Ajouter un enregistrement dans la table [Clients]
Jusque (OK = 0) ` Jusqu'à ce que l'utilisateur annule
```

(2) L'exemple suivant permet de rechercher un client dans la base. Le déroulement de la méthode dépend du résultat de la recherche. Si aucun client n'a été trouvé, l'utilisateur est autorisé à créer un nouveau client à l'aide de la commande AJOUTER ENREGISTREMENT. Si au moins un client a été trouvé, le premier enregistrement est affiché pour modification, à l'aide de la commande MODIFIER ENREGISTREMENT :

```
LECTURE ECRITURE ([Clients])
FORMULAIRE ENTREE ([Clients]; "Entrée1") `  Désigner le formulaire entrée
vlClientNo:=Num(Demander ("Saisissez un numéro de client :")
    ` On récupère le numéro du client
Si (OK =1)
    CHERCHER ([Clients]; [Clients]ClientNo = vlClientNo) ` Recherche du client
    Si (Enregistrements trouves([Clients]) = 0) ` Si aucun client n'a été trouvé…
        AJOUTER ENREGISTREMENT([Clients]) `Ajout d'un nouveau client
    Sinon
        Si(Non(Enregistrement verrouille([Clients])))
            MODIFIER ENREGISTREMENT([Clients]) `Modifier l'enregistrement
            LIBERER ENREGISTREMENT([Clients])
        Sinon
            ALERTE("Cet enregistrement est en train d'être modifié.")
        Fin de si
    Fin de si
Fin de si
```

Variables et ensembles système

La variable système OK prend la valeur 1 si l'enregistrement est validé et 0 s'il est annulé.

Référence

CREER ENREGISTREMENT, MODIFIER ENREGISTREMENT, NE PAS VALIDER, STOCKER ENREGISTREMENT, VALIDER.

AJOUTER SOUS ENREGISTREMENT (sousTable; formulaire{; *})

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table à utiliser pour la saisie des données
formulaire	Alpha	→	Formulaire à utiliser
*	*	→	Masquer les barres de défilement

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

AJOUTER SOUS ENREGISTREMENT permet d'ajouter un nouveau sous-enregistrement à sousTable, en utilisant formulaire. AJOUTER SOUS ENREGISTREMENT crée un nouveau sous-enregistrement en mémoire, en fait le sous-enregistrement courant et affiche formulaire. Il doit exister un enregistrement courant pour la table parente. S'il n'existe pas d'enregistrement de la table parente dans le process, AJOUTER SOUS ENREGISTREMENT ne fait rien. Le formulaire doit appartenir à sousTable.

Le sous-enregistrement reste en mémoire et sera sauvegardé si l'utilisateur clique sur un bouton de validation, appuie sur la touche **Entrée** ou si la commande VALIDER est exécutée. Dès que le sous-enregistrement a été ajouté ou modifié, l'enregistrement parent doit être sauvegardé pour que le sous-enregistrement soit sauvegardé.

Le sous-enregistrement n'est pas sauvegardé si l'utilisateur clique sur un bouton d'annulation, appuie sur les touches d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou si la commande NE PAS VALIDER est exécutée.

Après un appel à AJOUTER SOUS ENREGISTREMENT, la variable système OK prend la valeur 1 si le sous-enregistrement a été validé, sinon elle prend la valeur 0.

Le formulaire est affiché dans la fenêtre de premier plan du process, avec des barres de défilement et une case de redimensionnement. Si vous spécifiez le paramètre optionnel astérisque (*), la fenêtre sera dessinée sans les barres de défilement ni la case de redimensionnement.

Exemple

L'exemple suivant fait partie d'une méthode. Ces lignes de code ajoutent un sous-enregistrement pour un nouvel enfant dans l'enregistrement d'un employé. Les données pour l'enfant sont stockées dans la sous-table [Employés]Enfants. Notez que l'enregistrement de la table [Employés] doit être sauvegardé pour que le sous-enregistrement le soit également :

```
AJOUTER SOUS ENREGISTREMENT ([Employés]Enfants; "AjouterEnfant")
Si (OK = 1)  ` Si l'utilisateur a validé le sous-enregistrement
   STOCKER ENREGISTREMENT ([Employés])  ` Stocker l'enregistrement parent
Fin de si
```

Variables et ensembles système

Si le sous-enregistrement est validé, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Référence

CREER SOUS ENREGISTREMENT, MODIFIER SOUS ENREGISTREMENT, NE PAS VALIDER, STOCKER ENREGISTREMENT, SUPPRIMER SOUS ENREGISTREMENT, VALIDER.

Ancien (leChamp) → Expression

Paramètre	Type		Description
leChamp	Champ	→	Champ dont vous voulez obtenir l'ancienne valeur
Résultat	Expression	←	Valeur originale de champ

Description

La commande Ancien retourne la valeur qui était stockée dans champ avant qu'il n'ait été modifié par programmation ou pendant la saisie de données.

A chaque fois que vous changez d'enregistrement courant pour une table, 4D crée et maintient en mémoire un double de l'"image" du nouvel enregistrement courant au moment où il est chargé. Lorsque vous modifiez un enregistrement, vous travaillez avec l'image réelle de l'enregistrement, et non son double. Ce double est effacé lorsque que vous changez à nouveau d'enregistrement courant.

Ancien retourne la valeur de leChamp telle qu'elle est stockée dans le double de l'enregistrement. Autrement dit, pour un enregistrement existant, Ancien retourne la valeur du champ telle qu'elle avait été sauvegardée sur disque. Pour un enregistrement qui vient d'être créé, Ancien retourne la valeur vide par défaut correspondant au type de champ. Par exemple, si champ est de type Alpha, Ancien retourne une chaîne vide. Si champ est de type numérique, Ancien retourne zéro (0), etc.

Ancien fonctionne avec champ de la même manière, que le champ ait été modifié par programmation ou suite à des modifications effectuées par un utilisateur.

La fonction accepte tous les types de champs.

Pour restaurer la valeur originale d'un champ, assignez-lui la valeur retournée par Ancien.

Référence

Modifie.

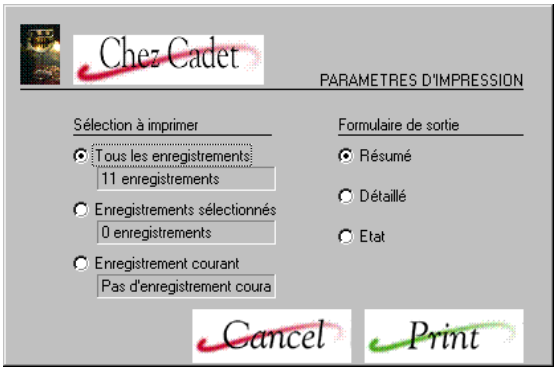
DIALOGUE ({table; }formulaire{; *})

Paramètre	Type		Description
table	Table	→	Table à laquelle appartient le formulaire Si omis : Table par défaut ou utilisation d'un formulaire projet
formulaire	Chaîne	→	Nom du formulaire table ou projet à afficher comme dialogue
*	*	→	Utiliser le même process

Description

La commande DIALOGUE présente le formulaire à l'utilisateur. Cette commande est souvent utilisée pour récupérer dans des variables des données fournies par l'utilisateur, ou pour lui présenter différentes informations, comme un choix d'options pour effectuer une opération.

Il est courant d'afficher le formulaire dans une fenêtre modale créée à l'aide de la commande Creer fenetre.
Voici un exemple typique de boîte de dialogue pouvant être affichée avec la commande DIALOGUE :



Utilisez DIALOGUE plutôt que CONFIRMER, ALERTE ou Demander lorsque les informations à afficher ou à recueillir sont plus complexes que celles que peuvent gérer ces trois autres commandes.

Note : Dans les bases de données converties, il est possible d'interdire la saisie dans les champs dans les boîtes de dialogue (et donc de limiter la saisie aux seules variables) via une option des Préférences de 4D (page Compatibilité). Cette restriction correspond au fonctionnement des anciennes versions de 4D.

A la différence d'AJOUTER ENREGISTREMENT et de MODIFIER ENREGISTREMENT, DIALOGUE n'utilise pas le formulaire entrée courant. Vous devez spécifier, dans le paramètre formulaire, le formulaire (formulaire projet ou formulaire table) à utiliser. De même, aucun ensemble de boutons n'est placé par défaut s'ils sont omis dans le formulaire. Dans ce cas, seule la touche **Echap** (Windows) ou **Esc** (Mac OS) permet de quitter le formulaire.

Le dialogue est validé si l'utilisateur clique sur le bouton de validation ou appuie sur la touche **Entrée**, ou si la commande VALIDER est exécutée.
A noter que la validation n'entraîne pas la sauvegarde : si le dialogue comporte des champs, vous devez appeler explicitement la commande STOCKER ENREGISTREMENT pour stocker les données éventuellement modifiées.

Le dialogue est annulé si l'utilisateur clique sur le bouton d'annulation, appuie sur la touche d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou si la commande NE PAS VALIDER est exécutée.

Si vous passez le paramètre facultatif ***** , le formulaire est chargé et affiché dans la dernière fenêtre ouverte du process courant et la commande termine son exécution en laissant le formulaire actif à l'écran.
Ce formulaire réagit alors "normalement" aux actions de l'utilisateur et est fermé lorsque du code 4D lié au formulaire (méthode objet ou méthode formulaire) appelle la commande NE PAS VALIDER ou VALIDER. Si le process courant se termine, les formulaires créés de cette façon sont automatiquement fermés en simulant un NE PAS VALIDER. Ce mode d'ouverture est particulièrement utile pour afficher une palette flottante en rapport avec un document, sans pour autant nécessiter un autre process.

Note : Vous devez créer une fenêtre avant d'appeler l'instruction DIALOGUE(form;*), il n'est pas possible d'utiliser la fenêtre du dialogue en cours dans le process ni la fenêtre créée par défaut pour chaque process. Dans le cas contraire, l'erreur -9909 est générée.

Après un appel à DIALOGUE, la variable système OK prend la valeur 1 si le dialogue est validé, et 0 sinon.

Exemples

(1) L'exemple suivant illustre l'utilisation de la commande DIALOGUE pour spécifier des critères de recherche. Un formulaire personnalisé contenant les variables vNom et vPays permet à l'utilisateur de saisir ses critères :

```
Creer fenetre (10; 40; 370; 220) ` Créer une fenêtre modale
DIALOGUE ("Form Recherche") ` Afficher le dialogue de recherche
FERMER FENETRE ` Nous n'avons plus besoin de la fenêtre
Si (OK = 1) ` Si le dialogue est validé
    CHERCHER ([Société]; [Société]Nom = vNom; *)
    CHERCHER (& [Société]Payst = vPays)
Fin de si
```

(2) L'exemple suivant permet de créer une palette d'outils :

```
    `Affichage palette d'outils
$window_palette:=Creer fenetre formulaire("tools";Form fenêtre palette)
DIALOGUE("tools";*) `Rend la main immédiatement
    `Affichage fenêtre document principal
$window_document:=Creer fenetre formulaire("doc";Form fenêtre standard)
DIALOGUE("doc")
```

Variables et ensembles système

Si l'utilisateur valide le dialogue, la variable système OK prend la valeur 1, si le dialogue est annulé OK prend la valeur 0.

Référence

AJOUTER ENREGISTREMENT, Creer fenetre, NE PAS VALIDER, VALIDER.

version 2004 (Modifiée)

Note de compatibilité
Cette fonction est conservée pour des raisons de compatibilité uniquement car elle s'appuie sur l'ancienne gestion des cycles d'exécution, elle-même obsolète depuis la version 6 (cf. commandes Avant, Pendant, etc.). Dans la plupart des cas, il est fortement conseillé d'utiliser la fonction Evenement formulaire et de tester si elle retourne l'événement Sur données modifiées.

Modifie (champ) → Booléen

Paramètre	Type		Description
champ	Champ	→	Champ dont vous voulez tester la modification
Résultat	Booléen	←	Vrai si une nouvelle valeur a été assignée au champ, sinon Faux

Description

Modifie retourne Vrai si une valeur a été assignée par programmation au champ champ ou s'il a été modifié lors de la saisie de données. La commande Modifie ne fonctionne que lorsqu'elle est appelée dans le cadre d'une méthode formulaire (ou d'une sous-méthode appelée par la méthode formulaire).

Attention, cette commande ne retourne une valeur significative qu'à l'intérieur d'un même cycle d'exécution. Elle est notamment remise à Faux pour tous les événements formulaires correspondant à l'ancien cycle d'exécution Pendant.

Dans le cas de la saisie de données, un champ est considéré comme modifié à partir du moment où un utilisateur l'édite (et change ou non sa valeur originale) puis le quitte pour un autre champ ou pour cliquer sur un objet de formulaire. Notez que le fait qu'un utilisateur active puis quitte un champ à l'aide de la touche Tabulation ne suffit pas en soi à ce que Modifie retourne Vrai. Le champ doit avoir été réellement modifié pour que Modifie retourne Vrai.

Dans le cas de l'exécution d'une méthode, un champ est considéré comme modifié si une valeur lui a été assignée (différente ou non de sa valeur précédente).

Note : La commande Modifie retourne toujours Vrai après l'exécution des commandes EMPILER ENREGISTREMENT et DEPILER ENREGISTREMENT.

Dans tous les cas, pour savoir si la valeur d'un champ a été effectivement modifiée, utilisez la commande Ancien.

Note : Bien que la fonction Modifie puisse être appliquée à tout type de champ, si vous l'utilisez conjointement avec la fonction Ancien, vous devez dans ce cas tenir compte des restrictions liées à cette fonction. Reportez-vous à la description de la commande Ancien.

Pendant la saisie de données, il est généralement plus pratique d'effectuer des opérations dans des méthodes objet que d'utiliser la fonction Modifie dans des méthodes formulaires. Comme une méthode objet reçoit l'événement Sur données modifiées à chaque fois qu'un champ est modifié, utiliser une telle méthode équivaut à appeler Modifie dans une méthode formulaire.

Exemples

(1) L'exemple suivant teste si le champ [Commandes]Quantité ou le champ [Commandes]Prix a été modifié. Si c'est le cas, le champ [Commandes]Total est recalculé :

```
Si ((Modifie ([Commandes]Quantité) | (Modifie ([Commandes]Prix))
    [Commandes]Total := [Commandes]Quantité * [Commandes]Prix
Fin de si
```

Notez que le même résultat aurait pu être obtenu en utilisant la seconde ligne de cette méthode en tant que méthode objet des champs [Commandes]Quantité et [Commandes]Prix dans le cadre de l'événement formulaire Sur données modifiées.

(2) Vous sélectionnez un enregistrement de la table [uneTable], puis vous appelez plusieurs sous-routines qui sont susceptibles de modifier le champ [uneTable]Champ important mais sans provoquer de sauvegarde de l'enregistrement. A la fin de la méthode principale, vous pouvez utiliser la commande Modifie pour déterminer si vous devez stocker l'enregistrement :

```
    ` L'enregistrement a été sélectionné comme enregistrement courant
    ` Puis vous effectuez des actions à l'aide des sous-routines
FAIRE QUELQUE CHOSE
FAIRE AUTRE CHOSE
NE PAS OUBLIER DE FAIRE CA
    ` ...
    ` Enfin, vous testez le champ pour déterminer s'il faut stocker l'enregistrement
Si (Modifie([uneTable]Champ important))
    STOCKER ENREGISTREMENT([uneTable])
Fin de si
```

Référence

Ancien, Evenement formulaire.

MODIFIER ENREGISTREMENT ({table}{; }{*})

Paramètre	Type		Description
table	Table	→	Table dans laquelle modifier des données ou Table par défaut si ce paramètre est omis
*		→	Cacher les barres de défilement

Description

La commande MODIFIER ENREGISTREMENT permet à l'utilisateur de modifier l'enregistrement courant de table, ou de la table par défaut si ce paramètre est omis. MODIFIER ENREGISTREMENT charge depuis le disque l'enregistrement courant pour le process en cours (s'il n'est pas déjà chargé par un autre utilisateur/process) et l'affiche dans le formulaire entrée courant. S'il n'y a pas d'enregistrement courant, MODIFIER ENREGISTREMENT ne fait rien. MODIFIER ENREGISTREMENT ne change pas la sélection courante.

Le formulaire est affiché dans la fenêtre se trouvant au premier plan du process. Elle comporte des barres de défilement et une case de contrôle de taille. Si vous passez le paramètre optionnel astérisque (*), ces éléments n'apparaîtront pas.

Pour que vous puissiez utiliser MODIFIER ENREGISTREMENT, l'enregistrement courant doit être en Lecture/écriture et ne doit pas être verrouillé.

Si le formulaire comporte des boutons de navigation parmi les enregistrements de la sélection, ils restent utilisables, ce qui permet à l'utilisateur de modifier des enregistrements puis de se déplacer pour en modifier d'autres.

L'enregistrement est sauvegardé si l'utilisateur clique sur un bouton du type Valider ou appuie sur la touche **Entrée**, ou encore si la commande VALIDER est exécutée.

L'enregistrement n'est pas sauvegardé si l'utilisateur clique sur un bouton du type **Annuler** ou appuie sur la touche d'annulation (**Echap** sous Windows, **Esc** sous Mac OS), ou encore si la commande NE PAS VALIDER est exécutée.

Après un appel à MODIFIER ENREGISTREMENT, la variable système OK prend la valeur 1 si l'enregistrement est validé, et 0 lorsqu'il est annulé.

Note : Même lorsqu'il est annulé, l'enregistrement reste en mémoire et peut être sauvegardé par la commande STOCKER ENREGISTREMENT si celle-ci est appelée avant que le pointeur d'enregistrement courant ne soit modifié.

Dans le cadre d'un MODIFIER ENREGISTREMENT, si l'utilisateur n'effectue aucune modification dans l'enregistrement et le valide, l'enregistrement ne sera pas considéré comme modifié et ne sera pas sauvegardé une nouvelle fois. Les actions telles que le changement de la valeur d'une variable, la sélection de cases à cocher ou de boutons radio ne sont pas qualifiées de modifications. Seule la modification de la valeur d'un champ, par le biais d'une saisie manuelle ou d'une méthode, provoque une nouvelle sauvegarde de l'enregistrement.

Exemple

Reportez-vous au second exemple de la commande AJOUTER ENREGISTREMENT.

Variables et ensembles système

La variable système OK prend la valeur 1 lorsque l'enregistrement est validé et 0 lorsqu'il est annulé. OK ne prend une valeur qu'après que l'enregistrement ait été effectivement validé ou annulé.

Référence

AJOUTER ENREGISTREMENT, Enregistrement modifie, Enregistrement verrouille, LECTURE ECRITURE, LIBERER ENREGISTREMENT.

MODIFIER SOUS ENREGISTREMENT (sousTable; formulaire{; *})

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table à utiliser pour la saisie de données
formulaire		→	Formulaire à utiliser pour la saisie
*		→	Cacher les barres de défilement

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

La commande MODIFIER SOUS ENREGISTREMENT affiche le sous-enregistrement courant de sousTable dans le formulaire en mode modification.

La table parente doit disposer d'un enregistrement courant. S'il n'y a pas d'enregistrement parent courant pour le process, MODIFIER SOUS ENREGISTREMENT ne fait rien. De plus, s'il n'y a pas de sous-enregistrement courant, MODIFIER SOUS ENREGISTREMENT ne fait rien non plus.

Les éventuelles modifications effectuées dans le sous-enregistrement ne seront effectivement sauvegardées que si l'enregistrement parent est lui-même sauvegardé.

Après un appel à MODIFIER SOUS ENREGISTREMENT, si le sous-enregistrement est validé, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Le formulaire est affiché dans la fenêtre de premier plan du process avec des barres de défilement et une case de redimensionnement. Si vous spécifiez l'astérisque optionnel (*), la fenêtre sera dessinée sans les barres de défilement ni la case de redimensionnement.

Variables et ensembles système

Si le sous-enregistrement est validé, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Référence

AJOUTER SOUS ENREGISTREMENT, NE PAS VALIDER, STOCKER ENREGISTREMENT, VALIDER.

51

Sauvegarde

Fichier historique

Fichier historique → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom long du fichier d'historique de la base

Description

La commande Fichier historique retourne le nom long (c'est-à-dire le chemin d'accès complet au fichier, y compris son nom) du fichier d'historique courant de la base ouverte.

Si la base fonctionne sans fichier d'historique, la fonction retourne une chaîne vide et la variable système OK prend la valeur 0.

Si la base fonctionne avec un fichier d'historique, la variable système OK prend la valeur 1. Le chemin d'accès retourné par la commande est exprimé avec la syntaxe de la plate-forme courante.

ATTENTION : Si vous exécutez cette commande depuis un poste 4D Client, seul le nom du fichier d'historique est retourné, pas le nom long.

Référence

FIXER FICHIER HISTORIQUE.

Variables et ensembles système

Si la base fonctionne sans fichier d'historique, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

FIXER FICHIER HISTORIQUE

FIXER FICHIER HISTORIQUE (historique | *)

Paramètre	Type		Description
historique | *	Chaîne | *	→	Nom du fichier d'historique ou * pour refermer l'historique courant

Description

La commande FIXER FICHIER HISTORIQUE crée ou ferme le fichier d'historique de la base de données, suivant la valeur que vous passez dans historique.

Note : Appeler la commande FIXER FICHIER HISTORIQUE équivaut à sélectionner/désélectionner l'option **Utiliser le fichier d'historique...** dans la page **Sauvegarde/Configuration** des Préférences de l'application.

Passez dans historique le nom ou le chemin d'accès complet du fichier d'historique à créer. Si vous passez uniquement un nom, le fichier sera créé dans le dossier "Logs" de la base, à côte du fichier de structure de la base. Si vous passez une chaîne vide, FIXER FICHIER HISTORIQUE présente une boîte de dialogue standard d'enregistrement de fichier, permettant à l'utilisateur de choisir le nom et l'emplacement du fichier d'historique à créer. Si le fichier est correctement créé, la variable OK prend la valeur 1. Autrement, si l'utilisateur clique sur le bouton **Annuler** ou si le fichier d'historique ne peut pas être créé, OK prend la valeur 0.

Note : Le nouveau fichier d'historique n'est pas généré immédiatement après l'exécution de la commande, mais après la prochaine sauvegarde (le paramètre est conservé dans le fichier de données, il sera pris en compte même si la base est refermée entre-temps. Vous pouvez appeler la commande SAUVEGARDER pour provoquer la création du fichier d'historique.

Si vous passez * dans historique, FIXER FICHIER HISTORIQUE referme le fichier d'historique courant de la base. La variable OK prend la valeur 1 lorsque le fichier d'historique est refermé.

Si vous utilisez FIXER FICHIER HISTORIQUE pour créer un fichier d'historique avant qu'une sauvegarde n'ait été réalisée et si le fichier de données contient déjà des enregistrements, 4D génère l'erreur -4447, que vous pouvez intercepter avec une méthode installée par APPELER SUR ERREUR.

Référence

APPELER SUR ERREUR.

Variables et ensembles système

OK prend la valeur 1 si le fichier d'historique est correctement créé ou fermé.

Gestion des erreurs

L'erreur -4447 est générée si l'opération ne peut pas être réalisée car la base de données doit être auparavant sauvegardée.

INTEGRER FICHIER HISTORIQUE

INTEGRER FICHIER HISTORIQUE (cheminAccès)

Paramètre	Type		Description
cheminAccès	Texte	→	Nom ou chemin d'accès du fichier d'historique à intégrer

Note préliminaire : Cette commande ne fonctionne qu'avec 4D Server. Elle ne peut être exécutée que via la commande Executer sur serveur ou dans une procédure stockée.

Description

La commande INTEGRER FICHIER HISTORIQUE provoque l'intégration dans la base courante du fichier d'historique dont le nom ou le chemin d'accès complet a été passé dans le paramètre cheminAccès. A l'issue de l'intégration, le fichier intégré devient le nouveau fichier d'historique courant de la base. Cette commande est destinée à la mise en place d'un système de sauvegarde par miroir logique (cf. section "Mise en place d'un miroir logique" dans le *Manuel de référence* de 4D Server).

Seuls les fichiers d'historique non archivés (extension .journal) peuvent être intégrés par cette commande. Aucune boîte de dialogue n'apparaît, toutefois une barre de progression est affichée à l'écran.

Vous pouvez passer dans cheminAccès un chemin d'accès absolu ou relatif au dossier de la base. Si vous passez une chaîne vide dans ce paramètre, une boîte de dialogue standard d'ouverture de fichier s'affiche, permettant de désigner le fichier à intégrer. Si la boîte de dialogue est annulée, aucun fichier n'est intégré et la variable système OK prend la valeur 0.

Lors de l'utilisation de cette commande, il est du ressort du développeur :
• d'installer la base miroir sur le poste miroir et de s'assurer que le fichier de données ne sera pas modifié autrement que par l'intégration d'un fichier d'historique via la commande INTEGRER FICHIER HISTORIQUE. Pour détecter qu'il s'agit de la version miroir de la base, il est possible de placer un fichier dans le dossier 4D Extensions ou le dossier de la base et de tester sa présence par exemple au cours de la Méthode base Sur ouverture. Si le fichier est présent, le mode miroir est activé.
• de mettre en place un système de communication entre la base en exploitation et la base miroir afin d'organiser l'envoi et la réception des segments de fichier d'historique. Il est possible d'utiliser pour cela un Web service ou 4D Internet Commands.
• de gérer les éventuelles erreurs de transmission entre les deux bases.

Référence

Nouveau fichier historique.

Variables et ensembles système

Si l'intégration s'effectue correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

En cas d'anomalie, la commande génère une erreur que vous pouvez intercepter à l'aide de la commande APPELER SUR ERREUR. Si au moins un enregistrement est verrouillé dans la base, la commande ne fait rien et l'erreur 1420 est générée.

LIRE INFORMATION RESTITUTION

LIRE INFORMATION RESTITUTION (sélecteur; info1; info2)

Paramètre	Type		Description
sélecteur	Entier long	→	Type d'information à récupérer
info1	Date | Entier	←	Valeur 1 du sélecteur
info2	Heure | Chaîne	←	Valeur 2 du sélecteur

Description

La commande LIRE INFORMATION RESTITUTION permet de récupérer des informations relatives à la dernière restitution automatique de la base.

Passez dans le paramètre sélecteur le type d'information à récupérer. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "Sauvegarde" :

Constante	Type	Valeur
Date dernière restitution	Entier long	0
Statut dernière restitution	Entier long	2

Le type et le contenu des paramètres info1 et info2 dépendent de la valeur de sélecteur.
• Si sélecteur = 0 (Date dernière restitution), info1 retourne la date et info2 l'heure de la dernière restitution automatique de la base.
• Si sélecteur = 2 (Statut dernière restitution), info1 retourne le numéro et info2 le texte du statut de la dernière restitution automatique de la base.

Note : Cette commande ne tient pas compte des restitutions manuelles de la base.

Référence

RESTITUER.

LIRE INFORMATION SAUVEGARDE

LIRE INFORMATION SAUVEGARDE (sélecteur; info1; info2)

Paramètre	Type		Description
sélecteur	Entier long	→	Type d'information à récupérer
info1	Date | Entier	←	Valeur 1 du sélecteur
info2	Heure | Chaîne	←	Valeur 2 du sélecteur

Description

La commande LIRE INFORMATION SAUVEGARDE permet de récupérer des informations relatives à la dernière sauvegarde effectuée sur les données de la base.

Passez dans le paramètre sélecteur le type d'information à récupérer. Vous pouvez utiliser une des constantes suivantes, placées dans le thème "Sauvegarde" :

Constante	Type	Valeur
Date dernière sauvegarde	Entier long	0
Statut dernière sauvegarde	Entier long	2
Prochaine sauvegarde	Entier long	4

Le type et le contenu des paramètres info1 et info2 dépendent de la valeur de sélecteur.
• Si sélecteur = 0 (Date dernière sauvegarde), info1 retourne la date et info2 l'heure de la dernière sauvegarde.
• Si sélecteur = 2 (Statut dernière sauvegarde), info1 retourne le numéro et info2 le texte du statut de la dernière sauvegarde.
• Si sélecteur = 4 (Prochaine sauvegarde), info1 retourne la date et info2 l'heure de la prochaine sauvegarde prévue.

Référence

RESTITUER.

Méthode base Sur arrêt sauvegarde

version 11 (Modifiée)

La Méthode base Sur arrêt sauvegarde est appelée à chaque fois qu'une sauvegarde de la base vient de se terminer. Les causes de l'arrêt de la sauvegarde peuvent être la fin de la copie, l'interruption par l'utilisateur ou une erreur.
Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

La Méthode base Sur arrêt sauvegarde permet de vérifier que la sauvegarde s'est correctement déroulée. Elle reçoit dans le paramètre **$1** une valeur indiquant le statut de la sauvegarde à l'issue de son exécution :
• Si la sauvegarde s'est terminée normalement, **$1** vaut 0.
• Si la sauvegarde a été interrompue à la suite d'une erreur ou par l'utilisateur, **$1** est différent de 0. Si la sauvegarde a été stoppée par la Méthode base Sur démarrage sauvegarde (**$0 # 0**), **$1** retourne le code effectivement retourné dans le paramètre **$0**. Ce principe vous permet de mettre en place un système de gestion d'erreurs personnalisé.
Dans tous les cas, vous pouvez obtenir des informations sur l'erreur à l'aide de la commande LIRE INFORMATION SAUVEGARDE.

Note : Vous devez impérativement déclarer le paramètre **$1** (entier long) dans la méthode base :

```
C_ENTIER LONG($1)
```

Référence

Méthode base Sur démarrage sauvegarde, SAUVEGARDER.

Méthode base Sur démarrage sauvegarde

La Méthode base Sur démarrage sauvegarde est appelée à chaque fois qu'une sauvegarde de la base est sur le point d'avoir lieu (sauvegarde manuelle, sauvegarde automatique périodique ou via la commande SAUVEGARDER).
Tous les environnements 4D sont concernés : 4D (tous modes), 4D Server ainsi que les applications 4D compilées et fusionnées avec 4D Volume Desktop.

La Méthode base Sur démarrage sauvegarde permet de contrôler le déclenchement de la sauvegarde. Au sein de la méthode, vous devez retourner dans le paramètre $0 une valeur autorisant ou refusant la sauvegarde :
• si $0 = 0, vous autorisez la sauvegarde.
• si $0 # 0, vous n'autorisez pas la sauvegarde. L'opération est annulée et une erreur est retournée. Vous pouvez récupérer l'erreur à l'aide de la commande LIRE INFORMATION SAUVEGARDE.

Vous pouvez utiliser cette méthode base pour contrôler les conditions d'exécution de la sauvegarde (utilisateur, date de la dernière sauvegarde, etc.).

Note : Vous devez impérativement déclarer le paramètre **$0** (entier long) dans la méthode base :

```
C_ENTIER LONG($0).
```

Référence

Méthode base Sur arrêt sauvegarde, SAUVEGARDER.

Nouveau fichier historique

Nouveau fichier historique → Texte

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Texte	←	Chemin d'accès complet du fichier d'historique refermé

Note préliminaire : Cette commande ne fonctionne qu'avec 4D Server. Elle ne peut être exécutée que via la commande Executer sur serveur ou dans une procédure stockée.

Description

La commande Nouveau fichier historique referme le fichier d'historique courant, le renomme et en crée un nouveau avec le même nom et au même emplacement que le précédent. Cette commande est destinée à la mise en place d'un système de sauvegarde par miroir logique (cf. section "Mise en place d'un miroir logique" dans le Manuel de référence de 4D Server).

La commande retourne le nom complet (chemin d'accès+nom) du fichier d'historique refermé (appelé "segment"). Ce fichier est stocké au même emplacement que le fichier d'historique courant (spécifié dans la page Configuration des préférences de l'application, thème Sauvegarde). La commande n'effectue aucun traitement (compression, segmentation) sur le fichier sauvegardé. Aucune boîte de dialogue n'apparaît.

Le fichier est renommé avec les numéros de sauvegarde courants de la base et du fichier d'historique, sur le modèle suivant : NomBase[NumSvgde-NumSvgdeHisto].journal. Par exemple :
• si la base MaBase.4DD a été sauvegardée 4 fois, le dernier fichier de sauvegarde se nomme MaBase[0004].4BK. Le nom du premier "segment" de fichier d'historique sera donc MaBase[0004-0000].journal.
• si la base MaBase.4DD a été sauvegardée 3 fois et que le fichier d'historique a été sauvegardé 5 fois depuis, le nom de la 6e sauvegarde du fichier d'historique sera MaBase[0003-0005].journal.

Référence

INTEGRER FICHIER HISTORIQUE.

Gestion des erreurs

En cas d'anomalie, la commande génère une erreur que vous pouvez intercepter à l'aide de la commande APPELER SUR ERREUR.

RESTITUER

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande RESTITUER affiche une boîte de dialogue d'ouverture permettant à l'utilisateur de sélectionner une archive à restituer.

Cette commande est utile dans le cadre d'interfaces personnalisées pour la gestion des sauvegardes.

Note : Dans le cadre d'une application 4D compilée et fusionnée avec *4D Volume Desktop*, la commande RESTITUER provoque l'affichage d'une boîte de dialogue système standard d'ouverture de fichiers, proposant par défaut les fichiers d'extension "4BK".

Référence

LIRE INFORMATION RESTITUTION, SAUVEGARDER.

SAUVEGARDER

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande SAUVEGARDER déclenche la sauvegarde de la base de données avec les paramétrages de sauvegarde courants. Aucune boîte de dialogue de confirmation n'est affichée. Une fenêtre de progression apparaît à l'écran.

Les paramétrages de sauvegarde sont définis dans les Préférences de l'application. Ils sont également stockés dans le fichier Backup.XML situé dans le sous-dossier Preferences/Backup de la base.

La commande SAUVEGARDER appelle la Méthode base Sur démarrage sauvegarde au début de son exécution et la Méthode base Sur arrêt sauvegarde à la fin de son exécution.
Attention, du fait de ce mécanisme, la commande ne doit PAS être appelée depuis l'une de ces méthodes base.

4D Server : Lorsqu'elle est appelée depuis un poste client, la commande SAUVEGARDER est considérée comme une procédure stockée, elle est toujours exécutée sur le serveur.

Référence

LIRE INFORMATION SAUVEGARDE, Méthode base Sur démarrage sauvegarde, RESTITUER.

Variables et ensembles système

Si la sauvegarde se déroule correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

En cas d'incident, une erreur est générée, que vous pouvez intercepter à l'aide d'une méthode de gestion d'erreurs installée à l'aide de la commande APPELER SUR ERREUR.

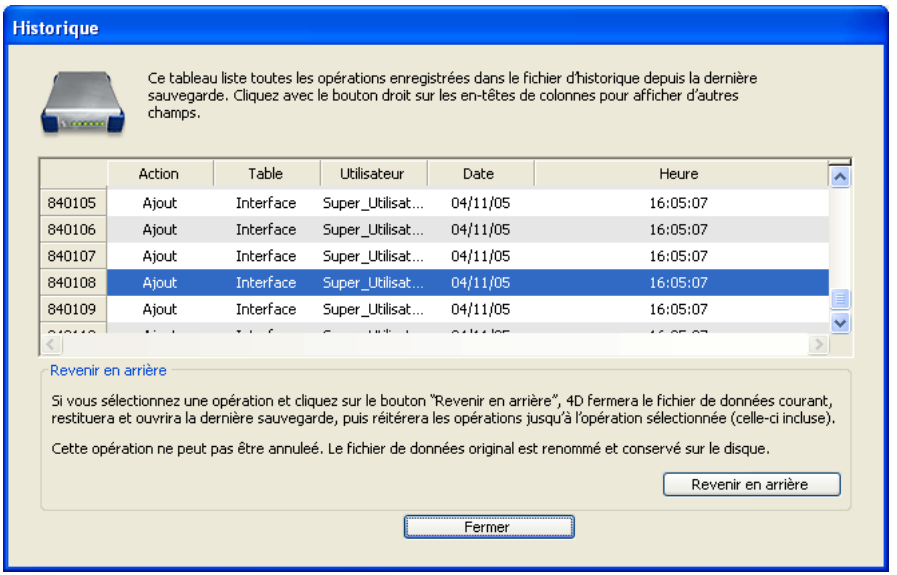
VERIFIER FICHIER HISTORIQUE

Paramètre **Type** **Description**

Cette commande ne requiert pas de paramètre

Description

La commande VERIFIER FICHIER HISTORIQUE affiche la boîte de dialogue de visualisation du fichier d'historique courant de la base (également accessible via la fenêtre du Centre de sécurité et de maintenance) :



Cette boîte de dialogue comporte le bouton **Revenir en arrière**, permettant d'annuler des opérations effectuées sur les données de la base. Pour plus d'informations sur cette boîte de dialogue, reportez-vous au manuel *Mode Développement* de 4D.

Note : La fonction de retour en arrière étant relativement puissante, il est conseillé de restreindre l'accès à la commande VERIFIER FICHIER HISTORIQUE aux administrateurs de la base.

Cette commande est utilisable dans le contexte d'une application monoposte uniquement. Elle permet notamment d'accéder à la fonction de retour en arrière depuis les applications *4D Volume Desktop* (applications sans mode Développement). Si elle est appelée dans une application client/serveur, elle ne fait rien et l'erreur 1421 est retournée.

Référence

Erreurs du gestionnaire de sauvegarde, RESTITUER.

Gestion des erreurs

• Si cette commande est exécutée dans une base de données fonctionnant sans fichier d'historique, elle ne fait rien et l'erreur 1403 est retournée.
• Si cette commande est exécutée sur une base client/serveur, elle ne fait rien et l'erreur 1421 est retournée.
Vous pouvez intercepter ces erreurs à l'aide d'une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

52

Sélections

ALLER A DERNIER ENREGISTREMENT

version 5

ALLER A DERNIER ENREGISTREMENT {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle vous voulez aller au dernier enregistrement ou Table par défaut si ce paramètre est omis

Description

ALLER A DERNIER ENREGISTREMENT désigne le dernier enregistrement de la sélection de table comme enregistrement courant et le charge en mémoire. Si la sélection est vide, ALLER A DERNIER ENREGISTREMENT ne fait rien.

Exemple

L'exemple suivant désigne le dernier enregistrement de la table [Contacts] comme enregistrement courant :

```
ALLER A DERNIER ENREGISTREMENT ([Contacts])
```

Référence

Avant selection, DEBUT SELECTION, ENREGISTREMENT PRECEDENT, ENREGISTREMENT SUIVANT, Fin de selection.

ALLER DANS SELECTION ({table; }position)

Paramètre	Type		Description
table	Table	→	Table dans laquelle aller à l'enregistrement spécifié ou Table par défaut si ce paramètre est omis
position	Numérique	→	Position de l'enregistrement dans la sélection

Description

La commande ALLER DANS SELECTION fait de l'enregistrement spécifié parmi la sélection courante de table l'enregistrement courant. La sélection courante n'est pas modifiée. Le paramètre position n'est pas équivalent au numéro retourné par Numero enregistrement. Ce paramètre représente la position de l'enregistrement au sein de la sélection courante. Cette position dépend de la manière dont la sélection a été créée et si elle a été triée.
Pour plus de précisions sur les numéros d'enregistrements, reportez-vous à la section A propos des numéros d'enregistrements.

S'il n'y a aucun enregistrement dans la sélection courante ou si position n'est pas dans la sélection, ALLER DANS SELECTION ne fait rien.

Si vous passez 0 dans position, il n'y a plus d'enregistrement courant dans table. Ce mécanisme permet de n'avoir plus aucun enregistrement sélectionné dans une liste, notamment dans les sous-formulaires inclus, lorsque le mode de sélection est "unique".

Exemple

L'exemple suivant charge les valeurs du champ [Personnes]Nom dans le tableau taNoms. Un tableau d'entiers longs, numEnr, est rempli avec des numéros qui représenteront ceux des enregistrements sélectionnés. Les deux tableaux sont alors triés :

```
    ` Créer ici la sélection de la table [Personnes]
    `...
    ` Récupérer les noms
SELECTION VERS TABLEAU ([Personnes]Nom; taNoms)
    ` Créer un tableau pour les numéros d'enregistrements sélectionnés
$vELNbEnrgs:=Taille tableau (taNoms)
TABLEAU ENTIER LONG (numEnr; $vELNbEnrgs)
```

```
Boucle ($Enrg; 1;$vELNbEnrgs)  ` Remplir le tableau avec ces numéros
    numEnr{$Enrg} := $Enrg
Fin de boucle
    ` Trier les deux tableaux par ordre alphabétique
TRIER TABLEAU (taNoms; numEnr; >)
```

Si le tableau taNoms est affiché dans une zone de défilement, l'utilisateur peut cliquer sur l'un des éléments. Comme les deux tableaux ont été triés de manière synchronisée, tout élément de numEnr fournit le numéro de l'enregistrement sélectionné pour lequel le nom a été stocké dans l'élément de taNoms correspondant.

La méthode objet de la zone de défilement taNoms suivante sélectionne le bon enregistrement dans la sélection de [Personnes] en fonction de ce que l'utilisateur a choisi dans la zone de défilement.

```
Au cas ou
    : (Evenement formulaire=Sur clic)
        Si (taNoms#0)
            ALLER DANS SELECTION (numEnr{taNoms})
        Fin de si
Fin de cas
```

Référence

Numero dans selection.

APPLIQUER A SELECTION (laTable; formule)

Paramètre	Type		Description
laTable	Table	→	Table dans laquelle appliquer la formule
formule	Formule	→	Ligne de code ou méthode

Description

La commande APPLIQUER A SELECTION applique formule à chaque enregistrement de la sélection courante de laTable. La formule peut être une ligne d'instructions ou une méthode (dans ce cas, le nom de la méthode doit être saisi sans ""). Si formule entraîne la modification d'un enregistrement de laTable, l'enregistrement modifié est sauvegardé. Si formule ne modifie pas d'enregistrement, aucune sauvegarde n'est réalisée. Si la sélection courante est vide, APPLIQUER A SELECTION ne fait rien. La formule peut faire appel à un champ d'une table liée si le lien est automatique.

La commande APPLIQUER A SELECTION peut être utilisée pour récupérer et traiter des informations sur une sélection d'enregistrements (par exemple, calcul d'un total), ou pour modifier une sélection (par exemple, mettre en majuscule la première lettre d'un champ). Si cette commande est utilisée à l'intérieur d'une transaction, toutes les opérations réalisées pourront être annulées si la transaction n'est pas validée.

4D Server : Le serveur n'exécute aucune des commandes passées dans formule. Chaque enregistrement de la sélection est renvoyé sur le poste client pour traitement et modification.

Un thermomètre de progression s'affiche pendant l'exécution d'un APPLIQUER A SELECTION. Un appel préalable à la commande SUPPRIMER MESSAGES permet de supprimer ce thermomètre. Lorsque le thermomètre de progression est affiché, l'utilisateur peut annuler l'opération.

Exemples

(1) L'exemple suivant met en majuscule tous les noms de la table :

```
APPLIQUER A SELECTION([Emp];[Emp]Nom:= Majusc([Emp]Nom))
```

(2) Lorsque APPLIQUER A SELECTION rencontre un enregistrement verrouillé et le modifie, celui-ci n'est pas sauvegardé. Tous les enregistrements verrouillés rencontrés par la commande sont placés dans un ensemble système appelé LockedSet. Après l'exécution d'un APPLIQUER A SELECTION, il est recommandé de tester l'ensemble LockedSet pour vérifier la présence d'enregistrements verrouillés. La boucle suivante s'exécute jusqu'à ce que tous les enregistrements aient été modifiés :

```
Repeter ` Pour chaque enregistrement verrouillé
    APPLIQUER A SELECTION ([Emp];[Emp]Nom:= Majusc([Emp]Nom))
    UTILISER ENSEMBLE ("LockedSet")
        ` Sélection des enregistrements verrouillés uniquement
        ` Jusqu'à ce qu'il n'y ait plus d'enregistrement verrouillé
Jusque (Enregistrements dans ensemble("LockedSet")=0)
```

(3) Cet exemple utilise une méthode :

```
TOUT SELECTIONNER([Emp])
APPLIQUER A SELECTION([Emp];Capitales)
```

Référence

EDITER FORMULE, Présentation des ensembles.

Variables et ensembles système

Si l'utilisateur clique sur le bouton Stop dans le thermomètre de progression, la variable système OK prend la valeur 0. Sinon, elle prend la valeur 1.

Avant selection {(table)} → Booléen

Paramètre	Type		Description
table	Table	→	Table pour laquelle vous testez si le pointeur se trouve avant la sélection
Résultat	Booléen	←	Avant sélection (Vrai) sinon (Faux)

Description

La fonction Avant selection retourne Vrai lorsque le pointeur d'enregistrement courant se trouve avant le premier enregistrement de la sélection courante de table. Avant selection est généralement utilisée pour vérifier si la commande ENREGISTREMENT PRECEDENT a déplacé le pointeur d'enregistrement courant avant le premier enregistrement. Si la sélection courante est vide, Avant selection retourne Vrai.

Pour replacer le pointeur d'enregistrement courant dans la sélection courante, utilisez les commandes DEBUT SELECTION, ALLER A DERNIER ENREGISTREMENT ou ALLER DANS SELECTION. ENREGISTREMENT SUIVANT ne replace pas le pointeur d'enregistrement courant dans la sélection courante.

Avant selection retourne Vrai dans l'en-tête lorsqu'un état est en cours d'impression à l'aide de la commande IMPRIMER SELECTION ou à partir de la commande de menu Imprimer. Vous pouvez utiliser le code suivant pour tester le premier en-tête et imprimer un en-tête spécial pour la première page :

```
    ` Méthode d'un formulaire sortie utilisé pour un état
$vpFormTable:=Table du formulaire courant
Au cas ou
        ` ...
    : (Evenement formulaire=Sur entête)
            ` La zone en-tête va être imprimée
        Au cas ou
            : (Avant selection($vpFormTable->))
                ` Le code pour la première rupture d'en-tête doit être placé ici
                ` ...
        Fin de cas
Fin de cas
```

Exemple

La méthode formulaire suivante est utilisée pendant l'impression d'un état. Elle définit une variable vTitre à imprimer dans la zone d'en-tête sur la première page :

```
   ` Méthode formulaire [Finances];"Tableau"
Au cas ou
      ` ...
   : (Evenement formulaire=Sur entête)
         ` La zone en-tête va être imprimée
      Au cas ou
         : (Avant selection([Finances]))
               ` Définir le titre pour la première page
            vTitre := "Etat des finances pour 1997"
         Sinon
            vTitre := ""  ` Effacer le titre pour les autres pages
      Fin de cas
Fin de cas
```

Référence

DEBUT SELECTION, ENREGISTREMENT PRECEDENT, Evenement formulaire, Fin de selection, IMPRIMER SELECTION.

DEBUT SELECTION

DEBUT SELECTION {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle charger le premier enregistrement de la sélection courantes ou Table par défaut si ce paramètre est omis

Description

DEBUT SELECTION charge en mémoire le premier enregistrement de la sélection courante de table et en fait l'enregistrement courant. Toutes les commandes de recherche, de sélection et de tri font également du premier enregistement l'enregistrement courant. Si la sélection courante est vide ou si l'enregistrement courant est déjà le premier enregistrement de la sélection, DEBUT SELECTION ne fait rien.

Cette commande est principalement utilisée après un appel à UTILISER ENSEMBLE, pour débuter une boucle dans la sélection d'enregistrements à partir du premier enregistrement. Cependant, il est tout à fait envisageable de l'appeler depuis une sous-routine lorsque vous souhaitez vous assurer que l'enregistrement est bien le premier.

Exemple

L'exemple suivant charge le premier enregistrement de la table [Clients] :

```
DEBUT SELECTION ([Clients])
```

Référence

ALLER A DERNIER ENREGISTREMENT, Avant selection, ENREGISTREMENT PRECEDENT, ENREGISTREMENT SUIVANT, Fin de selection.

ENREGISTREMENT PRECEDENT {(table)}

Paramètre	Type		Description
table	Table	→	Table dans laquelle se placer sur l'enregistrement précédent de la sélection courante ou Table par défaut si ce paramètre est omis

Description

ENREGISTREMENT PRECEDENT place le pointeur d'enregistrement courant sur l'enregistrement précédent dans la sélection courante de table pour le process courant. Si la sélection courante est vide, ou si Avant selection ou Fin de selection renvoie Vrai, ENREGISTREMENT PRECEDENT ne fait rien.

Si ENREGISTREMENT PRECEDENT place le pointeur d'enregistrement courant avant la sélection courante, Avant selection retourne Vrai, et il n'y a plus d'enregistrement courant. Dans ce cas, utilisez les commandes DEBUT SELECTION, ALLER A DERNIER ENREGISTREMENT ou ALLER DANS SELECTION pour replacer le pointeur d'enregistrement courant dans la sélection courante.

Référence

ALLER A DERNIER ENREGISTREMENT, Avant selection, DEBUT SELECTION, ENREGISTREMENT SUIVANT, Fin de selection.

ENREGISTREMENT SELECTION

ENREGISTREMENT SELECTION {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle réduire la sélection à un enregistrement

Description

La commande ENREGISTREMENT SELECTION réduit la sélection courante de table à l'enregistrement courant. S'il n'y a pas d'enregistrement courant ou si l'enregistrement courant n'est pas chargé en mémoire (cas particulier), ENREGISTREMENT SELECTION ne fait rien.

Note historique : A l'origine, cette commande était utile pour "replacer" dans la sélection courante un enregistrement qui avait été empilé puis dépilé de la pile d'enregistrements pendant que la sélection de la table était modifiée. Cependant, puisque depuis la version 6 de 4D, FIXER DESTINATION RECHERCHE vous permet d'effectuer une recherche sans changer la sélection ni l'enregistrement courants de la table, vous n'avez plus besoin d'empiler et de dépiler un enregistrement courant pour effectuer une recherche sur sa table. Par conséquent, ENREGISTREMENT SELECTION est moins utile, à moins que vous ne souhaitiez expressément, pour une autre raison, réduire la sélection d'une table à l'enregistrement courant.

ENREGISTREMENT SUIVANT

ENREGISTREMENT SUIVANT {(table)}

Paramètre	Type		Description
table	Table	→	Table dans laquelle se placer sur l'enregistrement suivant ou Table par défaut si ce paramètre est omis

Description

La commande ENREGISTREMENT SUIVANT place le pointeur d'enregistrement courant sur l'enregistrement suivant dans la sélection courante de table pour le process courant. Si la sélection courante est vide, ou si Avant selection ou Fin de selection retourne Vrai, ENREGISTREMENT SUIVANT ne fait rien.

Si ENREGISTREMENT SUIVANT place le pointeur d'enregistrement courant après la fin de la sélection courante, Fin de selection retourne Vrai, et il n'y a alors plus d'enregistrement courant. Lorsque Fin de selection retourne Vrai, utilisez les commandes DEBUT SELECTION, ALLER A DERNIER ENREGISTREMENT ou ALLER DANS SELECTION pour replacer le pointeur d'enregistrement courant dans la sélection courante.

Exemple

Reportez-vous à l'exemple de la commande VISUALISER SELECTION.

Référence

ALLER A DERNIER ENREGISTREMENT, Avant selection, DEBUT SELECTION, ENREGISTREMENT PRECEDENT, Fin de selection.

Enregistrements trouves {(table)} → Numérique

Paramètre	Type		Description
table	Table	→	Table dont vous souhaitez connaître le nombre d'enregistrements de la sélection courante ou Table par défaut si ce paramètre est omis
Résultat	Numérique	←	Nombre d'enregistrements dans la sélection courante de table

Description

Enregistrements trouves retourne le nombre d'enregistrements constituant la sélection courante de table (par opposition, Enregistrements dans table retourne le nombre total d'enregistrements d'une table).

Exemple

L'exemple suivant propose une technique de boucle couramment utilisée pour se déplacer parmi les enregistrements de la sélection courante. La même opération peut être réalisée à l'aide de la commande APPLIQUER A SELECTION :

```
DEBUT SELECTION ([Personnes])  ` Départ sur le premier enregistrement de la sélection
   ` Boucle une fois par enregistrement
Boucle ($VElEnreg; 1; Enregistrements trouves ([Personnes]))
   Faire quelque chose  ` Réaliser une opération avec l'enregistrement
   ENREGISTREMENT SUIVANT ([Personnes])  ` Passage à l'enregistrement suivant
Fin de boucle
```

Référence

Enregistrements dans table.

Fin de selection {(table)} → Booléen

Paramètre	Type		Description
table	Table	→	Table pour laquelle tester si le pointeur d'enregistrement courant est au-delà du dernier enregistrement de la sélection courante ou Table par défaut si ce paramètre est omis
Résultat	Booléen	←	Oui (Vrai), Non (Faux)

Description

La fonction Fin de selection retourne Vrai lorsque le pointeur de l'enregistrement courant se trouve après le dernier enregistrement de la sélection courante de table. Fin de selection est généralement utilisée pour tester si l'appel à la commande ENREGISTREMENT SUIVANT place ou non le pointeur d'enregistrement courant derrière le dernier enregistrement de la sélection. Si la sélection courante est vide, Fin de selection retourne Vrai.

Pour replacer le pointeur d'enregistrement courant dans la sélection, utilisez les commandes ALLER A DERNIER ENREGISTREMENT, DEBUT SELECTION ou ALLER DANS SELECTION. ENREGISTREMENT PRECEDENT ne replace pas le pointeur dans la sélection.

Fin de selection retourne également Vrai lors de l'impression du dernier pied de page d'un état, déclenchée par la commande IMPRIMER SELECTION ou le menu **Imprimer**. Vous pouvez utiliser l'instruction suivante pour intercepter le dernier pied de page et insérer une mention particulière :

```
      ` Méthode d'un formulaire sortie utilisé pour imprimer un état
$vpFormTable:=Table du formulaire courant
Au cas ou
            ` ...
      : (Evenement formulaire=Sur impression pied de page)
               ` Un pied
            Si (Fin de selection($vpFormTable->))
               ` Le code pour le dernier pied de page doit être placé ici
            Sinon
```

```
            ` Le code pour le pied de page doit être placé ici
        Fin de si
Fin de cas
```

Exemple

La méthode formulaire de l'exemple suivant est utilisée lors de l'impression d'un état. Elle crée la variable VPied, à imprimer dans le pied de page de la dernière page :

```
    ` Méthode formulaire [Finances];"Tableau"
Au cas ou
        ` ...
    : (Evenement formulaire=Sur impression pied de page)
        Si (Fin de selection([Finances]))
            VPied:= "©1997 SARL Dupont"   ` Définir le pied de page de la dernière page
        Sinon
            VPied:= ""   ` Effacer le pied de page pour toutes les autres pages
        Fin de si
Fin de cas
```

Référence

ALLER A DERNIER ENREGISTREMENT, Avant selection, ENREGISTREMENT SUIVANT, Evenement formulaire, IMPRIMER SELECTION.

LIRE ENREGISTREMENTS MARQUES

LIRE ENREGISTREMENTS MARQUES ({table; }nomEnsemble)

Paramètre	Type		Description
table	Table	→	Table de laquelle lire les enregistrements marqués Si omis, table du formulaire courant
nomEnsemble	Chaîne	→	Ensemble dans lequel stocker les enregistrements marqués

Description

La commande LIRE ENREGISTREMENTS MARQUES stocke dans l'ensemble désigné par le paramètre nomEnsemble les enregistrements marqués (c'est-à-dire, les enregistrements "surlignés" par l'utilisateur dans le formulaire liste) de la table passée en paramètre. Si le paramètre table est omis, la table du formulaire ou du sous-formulaire courant est utilisée.

En mode Développement ou dans le cadre de l'exécution des commandes VISUALISER SELECTION / MODIFIER SELECTION, cette commande peut être remplacée par l'appel de l'ensemble système UserSet, automatiquement maintenu par 4D. Toutefois, comme elle permet de désigner la table de laquelle récupérer les enregistrements marqués, la commande LIRE ENREGISTREMENTS MARQUES peut en outre gérer les sélections d'enregistrements dans les sous-formulaires inclus. En effet dans ce cas, les sélections des sous-formulaires pouvant provenir de tables différentes, l'ensemble système UserSet n'est pas géré par 4D. Pour plus d'informations sur l'ensemble UserSet, reportez-vous à la section Présentation des ensembles.

La commande LIRE ENREGISTREMENTS MARQUES peut être appelée hors du contexte d'un formulaire, cependant dans ce cas l'ensemble retourné est vide.
L'ensemble désigné par le paramètre nomEnsemble peut être local/client, process ou interprocess.

Note : Dans le cadre des sous-formulaires inclus, la commande LIRE ENREGISTREMENTS MARQUES retourne un ensemble vide si le sous-formulaire ne dispose pas de la propriété de sélection **Multilignes**. Dans ce contexte, pour connaître la ligne sélectionnée, vous devez utiliser la commande Numero dans selection.

Exemple

Cette méthode indique combien d'enregistrements sont sélectionnés dans le sous-formulaire affichant les enregistrements de la table [CDs] :

```
LIRE ENREGISTREMENTS MARQUES ([CDs];"$highlight")
ALERTE(Chaine(Enregistrements dans ensemble("$highlight"))"+" enregistrements
                                                                    sélectionnés.")
EFFACER ENSEMBLE("$highlight")
```

Référence

MARQUER ENREGISTREMENTS.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système *OK* prend la valeur 1, sinon elle prend la valeur 0.

MARQUER ENREGISTREMENTS ({table}{; nomEnsemble{; *}})

Paramètre	Type		Description
table	Table	→	Table de laquelle marquer les enregistrements Si omis, table du formulaire courant
nomEnsemble	Alpha	→	Ensemble d'enregistrements à marquer ou Ensemble Userset si ce paramètre est omis
*	*	→	Inactiver le défilement automatique de la liste

Description

La commande MARQUER ENREGISTREMENTS permet de "surligner" des enregistrements dans un formulaire en liste. Cette opération est identique à la sélection en mode liste, par l'utilisateur, d'enregistrement(s) à l'aide des combinaisons **Maj+clic** ou **Ctrl+clic** (Windows) ou **Commande+clic** (Mac OS). La sélection courante n'est pas modifiée.

Note : L'ensemble des enregistrements marqués est mis à jour après le redessinement des enregistrements, c'est-à-dire après la fin de l'exécution de toute la méthode d'appel — et non immédiatement après l'exécution de la commande MARQUER ENREGISTREMENTS.

Le paramètre table permet de désigner la table de laquelle les enregistrements doivent être "marqués". Ce paramètre permet en particulier de marquer les enregistrements des sous-formulaires inclus — n'appartenant donc pas à la table courante (cf. ci-dessous).

• Si vous passez un nom d'ensemble valide dans le paramètre nomEnsemble, la commande s'appliquera aux enregistrements de cet ensemble pour la table définie.
• Si vous omettez le paramètre nomEnsemble, la commande marquera les enregistrements de l'ensemble système UserSet courant. Cet ensemble est géré uniquement en mode Développement et dans le cadre de l'appel des commandes MODIFIER SELECTION / VISUALISER SELECTION). Si vous souhaitez marquer les enregistrements d'un sous-formulaire, vous devez passer un nom de table et d'ensemble. Pour plus d'informations sur l'ensemble UserSet, reportez-vous à la section Présentation des ensembles.

Le paramètre *, s'il est passé, provoque l'inactivation de la fonction de défilement automatique de la liste si les enregistrements marqués ne sont pas visibles. Ce mécanisme autorise la gestion personnalisée du défilement via la commande DEFILER LIGNES.

Note : Dans le cadre des sous-formulaires inclus, la commande MARQUER ENREGISTREMENTS ne fait rien si le sous-formulaire ne dispose pas de la propriété de sélection **Multilignes**. Dans ce contexte, pour marquer une ligne, vous devez utiliser la commande ALLER DANS SELECTION.

Exemple

Dans un formulaire en liste affiché par la commande MODIFIER SELECTION, vous souhaitez que l'utilisateur puisse effectuer des recherches, sans que la sélection courante soit modifiée. Pour cela, placez un bouton **Chercher** dans le formulaire et associez-lui la méthode suivante :

```
FIXER DESTINATION RECHERCHE(Vers ensemble;"UserSet")
CHERCHER
FIXER DESTINATION RECHERCHE(Vers sélection courante)
MARQUER ENREGISTREMENTS
```

Lorsque l'utilisateur clique sur le bouton, la boîte de dialogue standard de recherche apparaît. Une fois la recherche validée, les enregistrements trouvés sont surlignés, sans que la sélection courante ne soit modifiée.

Référence

DEFILER LIGNES, LIRE ENREGISTREMENTS MARQUES.

MODIFIER SELECTION ({table}{; modeSélection}{; saisieListe}{; *}{; *})

Paramètre	Type		Description
table	Table	→	Table à afficher et modifier ou Table par défaut si ce paramètre est omis
modeSélection	Entier long	→	Mode de sélection
saisieListe	Booléen	→	Autoriser saisie en liste
*		→	Utiliser formulaire sortie pour un seul enregistrement et cacher les barres de défilement dans le formulaire entrée
*		→	Afficher les barres de défilement dans le formulaire entrée (= annuler le second effet du premier paramètre *)

Description

La commande MODIFIER SELECTION est quasiment identique à la commande VISUALISER SELECTION. Reportez-vous à la commande VISUALISER SELECTION pour une description détaillée.
Les seules différences entre ces deux commandes sont les suivantes :

1. VISUALISER SELECTION et MODIFIER SELECTION provoquent l'affichage des enregistrements de la sélection courante de table dans le formulaire sortie courant, ou dans le formulaire entrée lorsque vous double-cliquez sur un enregistrement. Avec MODIFIER SELECTION, vous pouvez en plus modifier les champs de l'enregistrement dans le formulaire entrée lorsque vous double-cliquez dessus (s'il n'est pas déjà chargé par un autre utilisateur/process) ou en mode “Saisie en liste” (s'il est autorisé).

2. VISUALISER SELECTION charge les enregistrements en mode Lecture seulement dans le process courant, ce qui signifie qu'ils ne sont pas verrouillés en écriture pour les autres process. MODIFIER SELECTION place tous les enregistrements de la sélection en mode Lecture-écriture, ce qui signifie qu'ils sont automatiquement verrouillés en écriture pour les autres process. MODIFIER SELECTION libère les enregistrements lorsque son exécution est terminée.

Référence

Evenement formulaire, Présentation des ensembles, VISUALISER SELECTION.

Numero dans selection {(table)} → Numérique

Paramètre	Type		Description
table	Table	→	Table de laquelle retourner le numéro de l'enregistrement courant dans la sélection
Résultat	Numérique	←	Numéro dans la sélection

Description

Numero dans selection retourne la position de l'enregistrement courant dans la sélection courante de table.

Si la sélection est non vide et si l'enregistrement courant en fait partie, Numero dans selection retourne une valeur comprise entre 1 et Enregistrements trouves. Si la sélection est vide ou s'il n'y a pas d'enregistrement courant, Numero dans selection retourne 0.

Le numéro de l'enregistrement dans la sélection est différent du numéro retourné par Numero enregistrement (Numero enregistrement retourne le numéro physique de l'enregistrement dans la table). Le numéro de l'enregistrement dans la sélection dépend de la sélection courante.

Pour plus de précisions sur les numéros d'enregistrements, reportez-vous à la section A propos des numéros d'enregistrements.

Exemple

L'exemple suivant stocke le numéro de l'enregistrement courant de la sélection dans une variable :

```
` Obtenir le numéro de l'enregistrement dans la sélection
NumEnrCourant := Numero dans selection ([Personnes])
```

Référence

A propos des numéros d'enregistrements, ALLER DANS SELECTION, Enregistrements trouves.

Numero de ligne affichee → Entier long

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Entier long ←	Numéro de ligne en cours d'affichage

Description

La commande Numero de ligne affichee fonctionne uniquement dans le contexte de l'événement formulaire Sur affichage corps. Elle retourne le numéro de la ligne en cours de traitement durant l'affichage à l'écran d'une liste d'enregistrements ou des lignes d'une list box. Si Numero de ligne affichee est appelée en-dehors de l'affichage d'une liste ou d'une listbox, elle retourne 0.

Dans le cas d'une liste d'enregistrements, lorsque la ligne affichée n'est pas vide (c'est-à-dire lorsqu'elle est associée à un enregistrement), la valeur retournée par Numero de ligne affichee est identique à celle retournée par Numero dans selection.

Comme Numero dans selection, Numero de ligne affichee débute à 1. Cette commande est utile lorsque vous souhaitez appliquer un traitement à chaque ligne d'un formulaire liste ou d'une list box affiché(e) à l'écran, y compris aux lignes vides.

Exemple

Cet exemple permet d'appliquer une couleur alternée à un formulaire liste affiché à l'écran, même pour les lignes sans enregistrement :

```
      `Méthode du formulaire liste
Si (Evenement formulaire=Sur affichage corps)
   Si (Numero de ligne affichee % 2 = 0)
         `Noir sur blanc pour le texte des lignes paires
      FIXER COULEURS RVB([Table 1]Champ1; -1; 0x00FFFFFF)
   Sinon
         `Noir sur bleu pâle pour le texte des lignes impaires
      FIXER COULEURS RVB([Table 1]Champ1; -1; 0x00E0E0FF)
   Fin de si
Fin de si
```

Référence

Evenement formulaire, Numero dans selection.

REDUIRE SELECTION ({laTable; }nombre)

Paramètre	Type		Description
laTable	Table	→	Table de laquelle réduire la sélection ou Table par défaut si ce paramètre est omis
nombre	Numérique	→	Nombre d'enregistrements à conserver

Description

La commande REDUIRE SELECTION crée une nouvelle sélection d'enregistrements pour table. La commande réduit la sélection de laTable aux nombre premiers enregistrements. REDUIRE SELECTION s'applique à la sélection courante de laTable pour le process courant. Le premier enregistrement de la nouvelle sélection courante devient l'enregistrement courant.

Note : Si l'instruction REDUIRE SELECTION(0) est exécutée, il n'y a plus de sélection ni d'enregistrement courants dans la table.

Exemple

L'exemple suivant établit des statistiques pour une compétition mondiale parmi les revendeurs dans plus de 20 pays. Pour chaque pays, les trois meilleurs revendeurs qui ont vendu plus de 50 000 Euros de produits font partie des 100 meilleurs revendeurs dans le monde et sont recompensés. Avec peu de lignes de code, cette requête complexe peut être effectuée en utilisant des recherches indexées :

```
ENSEMBLE VIDE([Revendeurs];"Gagnants")  ` Créer un ensemble vide
SCAN INDEX([Revendeurs]Montant;100;<)  ` Chercher à la fin de l'index
NOMMER ENSEMBLE([Revendeurs];"100 Meilleurs Revendeurs")
        ` Placer les enregistrements sélectionnés dans un ensemble
Boucle ($Pays;1;Enregistrements dans table([Pays]))  ` pour chaque pays
        ` Chercher les revendeurs dans ce pays
    CHERCHER([Revendeurs];[Revendeurs]Pays=NomPays;*)
        ` ...qui ont vendu plus de 50000
    CHERCHER(&;[Revendeurs];[Revendeurs]Montant vendu>=50000)
    NOMMER ENSEMBLE([Revendeurs];"GagnantsRevendeurs")
        ` Les placer dans un ensemble
        ` Ils doivent être placés dans le groupe des 100 meilleurs revendeurs
```

```
        INTERSECTION("GagnantsRevendeurs";"100 Meilleurs Revendeurs";
                                                    "GagnantsRevendeurs")
        UTILISER ENSEMBLE("GagnantsRevendeurs") ` Gagnants potentiels pour le pays
            ` Trier les résultats en ordre décroissant
        TRIER([Revendeurs];[Revendeurs]Montant vendu;<)
        REDUIRE SELECTION([Revendeurs];3)  ` Garder les trois meilleurs
        NOMMER ENSEMBLE([Revendeurs];"GagnantsRevendeurs")
            ` Les gagnants pour le pays
            ` Les placer dans un ensemble des gagnants mondiaux
        REUNION("GagnantsRevendeurs";"LesGagnants";"LesGagnants")
    Fin de boucle
    EFFACER ENSEMBLE("100 Meilleurs Revendeurs")
        ` Nous n'avons plus besoin de cet ensemble
    EFFACER ENSEMBLE("GagnantsRevendeurs")  ` Nous n'avons plus besoin de cet ensemble
    UTILISER ENSEMBLE("LesGagnants")  ` Voici les gagnants
    EFFACER ENSEMBLE("LesGagnants")  ` Nous n'avons plus besoin de cet ensemble
    FORMULAIRE SORTIE([Revendeurs];"Lettre des gagnants")  ` Sélectionner la lettre
    IMPRIMER SELECTION([Revendeurs]) ` Imprimer les lettres
```

Référence

CHERCHER, Présentation des ensembles, SCAN INDEX, TRIER.

SCAN INDEX (leChamp; nombre{; > ou <})

Paramètre	Type		Description
leChamp	Champ	→	Champ indexé avec lequel "scanner" les enregistrements
nombre	Numérique	→	Nombre d'enregistrements à retourner
> ou <		→	> à partir du début de l'index < à partir de la fin de l'index

Description

La commande SCAN INDEX retourne une sélection de nombre enregistrements de la table du champ leChamp. Cette commande est extrêmement rapide car elle utilise l'index.
Si vous passez <, SCAN INDEX retourne nombre enregistrements à partir de la fin de l'index (valeurs supérieures). Si vous passez >, SCAN INDEX retourne nombre enregistrements à partir du début de l'index (valeurs inférieures). Si vous ne passez pas le dernier paramètre, la commande retourne nombre enregistrements à partir du début de l'index (équivaut à passer >).

Note : La sélection obtenue n'est pas triée.

SCAN INDEX fonctionne uniquement avec des champs indexés. Cette commande modifie la sélection courante de la table pour le process courant et charge le premier enregistrement de la sélection en tant qu'enregistrement courant.

Si vous spécifiez un nombre d'enregistrements supérieur au nombre d'enregistrements présents dans la table, SCAN INDEX retourne tous les enregistrements.

Exemple

L'exemple suivant envoie des lettres aux 50 plus mauvais clients puis aux 50 meilleurs clients :

```
SCAN INDEX([Clients]TotalDû;50;<) ` Obtenir la liste des 50 plus mauvais clients
TRIER([Clients]CodePostal;>) ` Trier par code postal
FORMULAIRE SORTIE([Clients];"Menace")
IMPRIMER SELECTION([Clients]) ` Imprimer les lettres
SCAN INDEX([Clients]TotalDû;50;>) ` Obtenir la liste des 50 meilleurs clients
TRIER([Clients]CodePostal;>) ` Trier par code postal
FORMULAIRE SORTIE([Clients];"Remerciement")
IMPRIMER SELECTION([Clients]) ` Imprimer les lettres
```

Référence

CHERCHER, REDUIRE SELECTION, TRIER.

SUPPRIMER SELECTION {(laTable)}

Paramètre	Type		Description
laTable	Table	→	Table de laquelle supprimer la sélection courante ou Table par défaut si ce paramètre est omis

Description

La commande SUPPRIMER SELECTION supprime la sélection courante d'enregistrements de laTable. Si la sélection courante est vide, SUPPRIMER SELECTION ne fait rien. Après la suppression des enregistrements, la sélection courante est vide. Les enregistrements supprimés pendant une transaction sont verrouillés pour les autres utilisateurs et/ou process jusqu'à ce que la transaction soit validée ou annulée.

Attention : La suppression d'une sélection d'enregistrements est une opération définitive. Elle ne peut être annulée par la suite.

L'option **Enregistrement(s) définitivement supprimé(s)** de l'Inspecteur des tables vous permet d'augmenter la vitesse des suppressions lors de l'utilisation de SUPPRIMER SELECTION.

Exemple

(1) L'exemple suivant affiche tous les enregistrements de la table [Personnes] et permet à l'utilisateur de sélectionner ceux qu'il souhaite effacer. L'exemple est en deux parties. La première est la méthode affichant les enregistrements. La seconde est la méthode objet d'un bouton 'Supprimer'. Voici la première méthode :

```
TOUT SELECTIONNER ([Personnes]) ` Sélection de tous les enregistrements
FORMULAIRE SORTIE ([Personnes]; "FormSortie")
    ` Définition du formulaire listant les enregistrements
VISUALISER SELECTION ([Personnes]) ` Affichage de tous les enregistrements
```

Voici la méthode objet du bouton Supprimer, apparaissant dans le pied de page du formulaire sortie. La méthode utilise les enregistrements sélectionnés par l'utilisateur (l'ensemble système UserSet) pour effacer la sélection (notez que si l'utilisateur ne sélectionne aucun enregistrement, SUPPRIMER SELECTION ne fait rien) :

```
    ` Demander confirmation que l'utilisateur veut réellement supprimer les enregistrements
CONFIRMER("Vous avez sélectionné "+Chaine(Enregistrements dans
                                    ensemble("UserSet"))+" enregistrements à supprimer."+
                                Caractere(13)+"Cliquez sur OK pour confirmer l'opération.")
Si(OK=1)
   UTILISER ENSEMBLE("UserSet") ` Utiliser l'ensemble défini par l'utilisateur
   SUPPRIMER SELECTION([Personnes]) ` Supprimer la sélection d'enregistrements
Fin de si
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
```

(2) Lorsqu'un SUPPRIMER SELECTION rencontre un enregistrement verrouillé, celui-ci n'est pas supprimé. Tous les enregistrements verrouillés sont placés dans un ensemble système nommé LockedSet. Après l'exécution de SUPPRIMER SELECTION, vous pouvez tester cet ensemble afin de vérifier si des enregistrements étaient verrouillés. La boucle suivante s'exécutera jusqu'à ce que tous les enregistrements aient été supprimés.

```
Repeter  ` Répéter pour chaque enregistrement verrouillé
   SUPPRIMER SELECTION ([CetteTable])
   Si (Enregistrements dans ensemble("LockedSet")#0)
         ` Si des enregistrements sont verrouillés
      UTILISER ENSEMBLE("LockedSet")   ` Sélectionner les enregistrements verrouillés
   Fin de si
Jusque (Enregistrements dans ensemble("LockedSet")=0)   ` Jusqu'à ce qu'il n'y en ait plus
```

Référence

MODIFIER SELECTION, Présentation des ensembles, Verrouillage d'enregistrements, VIDER TABLE, VISUALISER SELECTION.

TOUT SELECTIONNER {(table)}

Paramètre	Type		Description
table	Table	→	Table de laquelle vous voulez sélectionner tous les enregistrements ou Table par défaut si ce paramètre est omis

Description

La commande TOUT SELECTIONNER sélectionne tous les enregistrements de table pour le process courant. TOUT SELECTIONNER fait du premier enregistrement de la sélection l'enregistrement courant et le charge en mémoire. TOUT SELECTIONNER retourne les enregistrements dans l'ordre par défaut, qui est l'ordre dans lequel ils ont été stockés sur le disque.

Exemple

L'exemple suivant affiche tous les enregistrements de la table [Personnes] :

```
TOUT SELECTIONNER([Personnes])      ` Sélection de tous les enregistrements de la table
   `Affichage des enregistrements dans le formulaire sortie
VISUALISER SELECTION ([Personnes])
```

Référence

CHERCHER, Enregistrements dans table, Enregistrements trouves, MODIFIER SELECTION, TRIER, VISUALISER SELECTION.

VIDER TABLE {(laTable)}

Paramètre	Type		Description
laTable	Table	→	Table de laquelle vous voulez supprimer tous les enregistrements ou Table par défaut si ce paramètre est omis

Description

La commande VIDER TABLE supprime tous les enregistrements de laTable de façon très rapide. Après l'appel de la commande, il n'y a plus de sélection courante ni d'enregistrement courant.

L'effet de cette commande est semblable à celui d'une séquence TOUT SELECTIONNER / SUPPRIMER SELECTION, toutefois son fonctionnement diffère sur les points suivants :
• Le trigger éventuel n'est pas appelé.
• L'intégrité référentielle des données n'est pas contrôlée.
• Aucune transaction ne doit être en cours dans le process exécutant VIDER TABLE. Si c'est le cas, la commande ne fait rien et la variable système *OK* prend la valeur 0.
• Si un enregistrement au moins est verrouillé par un autre process, la commande échoue : une erreur est générée et la variable *OK* prend la valeur 0. L'ensemble système *LockedSet* n'est pas créé.
• Si laTable est déjà vide, VIDER TABLE ne fait rien et fixe la variable *OK* à 1.
• Si laTable est en lecture seule, VIDER TABLE ne fait rien et fixe la variable *OK* à 0.
• L'opération est enregistrée dans le fichier d'historique s'il est présent.

La commande VIDER TABLE est donc à manier avec précaution mais est très efficace pour, par exemple, supprimer rapidement des données temporaires.

Note : Le concept et le fonctionnement de cette commande sont proches de ceux de la commande TRUNCATE (TABLE) du SQL.

Référence

SUPPRIMER SELECTION.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

VISUALISER SELECTION ({table}{; modeSélection}{; saisieListe}{; *}{; *})

Paramètre	Type		Description
table	Table	→	Table à laquelle appartient la sélection ou Table par défaut si ce paramètre est omis
modeSélection	Entier long	→	Mode de sélection
saisieListe	Booléen	→	Autoriser saisie en liste
*		→	Utiliser le formulaire sortie en cas de sélection d'un seul enregistrement et masquer les barres de défilement dans le formulaire entrée
*		→	Afficher les barres de défilement dans le formulaire entrée (= annuler le second effet du premier paramètre *)

Description

VISUALISER SELECTION affiche, pour le process en cours, la sélection courante de table dans le formulaire sortie courant. Les enregistrements sont affichés sous la forme d'une liste que l'on peut faire défiler, semblable à celle du mode Développement. Lorsque l'utilisateur double-clique sur un enregistrement, par défaut celui-ci s'affiche dans le formulaire entrée courant. La liste est placée dans la fenêtre de premier plan.

Si vous souhaitez afficher une sélection et pouvoir également modifier un enregistrement dans le formulaire entrée courant une fois que vous avez double-cliqué dessus (comme vous le faites dans la fenêtre du mode Développement) ou via le mode “Saisie en liste”, utilisez MODIFIER SELECTION au lieu de VISUALISER SELECTION. Toutes les explications suivantes s'appliquent à ces deux commandes, hormis la possibilité de modifier des enregistrements.

Après qu'un VISUALISER SELECTION ait été exécuté, il n'y a plus d'enregistrement courant. Vous devez utiliser une commande telle que DEBUT SELECTION ou ALLER A DERNIER ENREGISTREMENT pour en récupérer un.

Le paramètre modeSélection vous permet de définir les possibilités de sélection d’enregistrements dans la liste à l'aide de la souris. Vous pouvez passer dans ce paramètre une des constantes suivantes du thème “Paramètres de formulaire” :
• si vous passez Pas de sélection (0), il ne sera pas possible de sélectionner un enregistrement dans la liste.

• si vous passez Sélection unique (1), seule la sélection d'un enregistrement à la fois sera autorisée.
• si vous passez Sélection multiple (2), l'utilisateur pourra sélectionner plusieurs enregistrements. Pour sélectionner des enregistrements contigus, il suffit de cliquer sur le premier enregistrement à sélectionner puis d'appuyer sur la touche **Majuscule** avant de cliquer sur le dernier. Pour sélectionner des enregistrements non adjacents, il suffit de cliquer sur chaque enregistrement en maintenant enfoncée la touche **Ctrl** (sous Windows) ou **Commande** (sous Mac OS).
Si vous ne passez pas le paramètre modeSélection, par défaut le mode "Sélection multiple" est utilisé.

Le paramètre saisieListe vous permet d'autoriser le mode "Saisie en liste" dans la liste affichée. Ce mode permet à l'utilisateur de sélectionner et de modifier directement les valeurs des enregistrements dans le formulaire sortie. Passez Vrai pour autoriser ce mode ou Faux pour ne pas l'autoriser. Par défaut, si vous ne passez pas le paramètre saisieListe, le mode "Saisie en liste" n'est pas autorisé.
A noter qu'avec la commande VISUALISER SELECTION, ce paramètre permet uniquement la sélection de valeurs dans la liste et non leur modification. En effet, la commande VISUALISER SELECTION charge les enregistrements de la sélection courante en Lecture seulement dans le process en cours. Seule la commande MODIFIER SELECTION permet effectivement la saisie de valeurs.

Note : La commande CHOIX SAISISSABLE permet d'activer ou de désactiver le mode Saisie en liste à la volée.

Lorsque la sélection ne contient qu'un enregistrement, et que le premier paramètre optionnel * n'est pas passé, l'enregistrement s'affichera directement dans le formulaire entrée. Si le premier paramètre optionnel * est spécifié, l'enregistrement unique sera affiché dans le formulaire sortie. Si le premier paramètre optionnel * est spécifié et que l'utilisateur affiche l'enregistrement dans le formulaire entrée en double-cliquant dessus, les barres de défilement du formulaire seront masquées. Pour annuler ce second effet du premier paramètre optionnel *, passez le second paramètre optionnel *.

Vous pouvez placer des boutons personnalisés dans la zone d'en-tête ou de pied de page du formulaire sortie pour terminer l'exécution de la commande VISUALISER SELECTION. Vous pouvez utiliser des boutons automatiques **Valider** ou **Annuler** permettant de sortir de la liste ou utiliser une méthode objet qui appelle les commandes VALIDER ou NE PAS VALIDER. Lorsqu'un formulaire sortie appelé par la commande VISUALISER SELECTION est dépourvu de boutons, seule la touche **Echap** (Windows) ou **Esc** (Mac OS) permet de quitter la liste.

Pendant et après l'exécution d'un VISUALISER SELECTION, les enregistrements sélectionnés par l'utilisateur sont conservés dans un ensemble système nommé UserSet. Après l'exécution de la commande, l'ensemble UserSet est accessible pendant un VISUALISER SELECTION aux méthodes objet de boutons, aux méthodes appelées par des commandes de menu, ainsi que pour la méthode projet qui avait appelé VISUALISER SELECTION.

Exemples

(1) L'exemple suivant sélectionne tous les enregistrements de la table [Personnes]. La commande VISUALISER SELECTION est alors utilisée pour afficher les enregistrements et permettre à l'utilisateur de désigner ceux qu'il souhaite imprimer. Enfin, les enregistrements sélectionnés sont récupérés à l'aide de la commande UTILISER ENSEMBLE et imprimés avec IMPRIMER SELECTION :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
VISUALISER SELECTION ([Personnes]; *)` Affichage des enregistrements
UTILISER ENSEMBLE ("UserSet")
    ` Utiliser uniquement les enregistrements sélectionnés par l'utilisateur
IMPRIMER SELECTION ([Personnes]) ` Imprimer les enregistrements sélectionnés
```

(2) Reportez-vous à l'exemple n°6 de la commande Evenement formulaire ; il indique tous les tests que vous pourrez avoir besoin d'effectuer pour surveiller la totalité des événements intervenant pendant l'exécution de la commande VISUALISER SELECTION.

(3) Pour reproduire, par exemple, les fonctionnalités apportées par le menu **Enregistrements** du mode Développement lorsque vous utilisez MODIFIER SELECTION ou VISUALISER SELECTION en mode Application, procédez de la manière suivante :

I. Dans le mode Développement, créez une barre de menus comportant les menus qui vous intéressent (par exemple Tout montrer, Recherche et Trier).
II. Associez cette barre de menus (à l'aide du menu “Barre de menus associée” dans la boîte de dialogue des propriétés du formulaire) au formulaire sortie utilisé avec les commandes VISUALISER SELECTION ou MODIFIER SELECTION.
III. Associez les méthodes projet suivantes à vos commandes de menu :

```
    ` M_TOUT_MONTRER (associée à la commande de menu Tout montrer)
$vpCourTable:=Table du formulaire courant
TOUT SELECTIONNER($vpCourTable->)
```

```
    ` M_Recherche (associée à la commande de menu Recherche)
$vpCourTable:=Table du formulaire courant
CHERCHER($vpCourTable->)

    ` M_TRIER (associée à la commande de menu Trier)
$vpCourTable:=Table du formulaire courant
TRIER($vpCourTable->)
```

Vous pouvez aussi utiliser d'autres commandes telles que IMPRIMER SELECTION, QR ETAT, etc. , afin de reproduire les commandes de menu "standard" à chaque fois que vous affichez ou modifiez une sélection en mode Application. Grâce à la commande Table du formulaire courant, ces méthodes sont génériques et les barres de menus auxquelles elles sont associées peuvent être rattachées à tout formulaire de sortie ou à toute table.

Référence

Evenement formulaire, MODIFIER SELECTION, Présentation des ensembles.

53

Sélections Temporaires

Présentation des Sélections Temporaires

Les sélections temporaires vous permettent de manipuler plusieurs sélections à la fois. Une sélection temporaire est une liste ordonnée d'enregistrements pour une table dans un process. Cette liste ordonnée d'enregistrements peut avoir un nom et est conservée en mémoire. Les sélections temporaires vous fournissent un moyen facile de garder en mémoire l'ordre et l'enregistrement courant de la sélection.

Les commandes suivantes vous permettent de travailler avec les sélections temporaires :

- COPIER SELECTION
- DEPLACER SELECTION
- UTILISER SELECTION
- EFFACER SELECTION
- CREER SELECTION SUR TABLEAU

Les sélections temporaires sont créées par les commandes COPIER SELECTION, DEPLACER SELECTION et CREER SELECTION SUR TABLEAU. Les sélections temporaires sont généralement utilisées pour travailler avec une ou plusieurs sélections, effectuer une sauvegarde puis retrouver une sélection ordonnée. Il peut y avoir plusieurs sélections temporaires pour chaque table dans un process. Pour réutiliser une sélection temporaire en tant que sélection courante, appelez UTILISER SELECTION. Lorsque vous en avez terminé avec une sélection temporaire, utilisez EFFACER SELECTION.

Note : La combinaison de l'instruction FIXER DESTINATION RECHERCHE(Vers sélection temporaire;selectiontemp) et d'une commande de recherche (par exemple CHERCHER) permet également de créer une sélection temporaire. Reportez-vous à la description de la commande FIXER DESTINATION RECHERCHE.

Les sélections temporaires peuvent avoir une portée (une aire d'action) locale, process ou interprocess.

Une sélection temporaire est locale lorsque son nom est précédé du symbole $. Elle est process lorsque son nom n'est précédé d'aucun symbole. Elle est interprocess lorsque son nom est précédé des symboles (<>) — le signe “inférieur à” suivi du symbole “supérieur à”.

Note : Cette syntaxe peut être utilisée sous Windows et Mac OS. Sous Mac OS, vous pouvez aussi utiliser le symbole "diamant" (Option + v sur un clavier français).

La portée d'une sélection temporaire interprocess est identique à celle d'une variable interprocess. On peut accéder à une sélection temporaire interprocess à partir de n'importe quel process.
Avec 4D en mode distant et 4D Server, une sélection temporaire interprocess n'est accessible que pour les process du client qui l'a créée. Une sélection temporaire interprocess n'est pas accessible aux autres clients.

Une sélection temporaire process n'est disponible que dans le process où elle a été créée et sur le serveur.

Une sélection temporaire locale est définie pour le process qui l'a créée et n'est pas visible sur le serveur.

Attention : Créer une sélection temporaire nécessite l'accès à la sélection de la table. Comme les sélections sont conservées sur le serveur et qu'un process local n'a pas accès à 4D Server, ne cherchez pas à utiliser des sélections temporaires dans un process local.

Visibilité des sélections temporaires
Le tableau suivant indique les principes de visibilité des sélections temporaires en fonction de leur portée et de leur lieu de création :

	Process client	Autres process du client	Autres clients	Process serveur	Autres process du serveur
Création dans un process client					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Création dans un process serveur					
$test				x	
test				x	
<>test				x	x

Sélections temporaires et ensembles
Voici les différences majeures entre les ensembles et les sélections temporaires :

• Une sélection temporaire est une liste ordonnée d'enregistrements, ce que n'est pas un ensemble.
• Les ensembles sont économes en mémoire car il n'ont besoin que d'un bit par enregistrement de la table. Les sélections temporaires ont besoin de 4 octets pour chaque enregistrement dans la sélection.

• A la différence des ensembles, les sélections temporaires ne peuvent pas être sauvegardées sur disque.
• Alors que les opérations standard Intersection, Reunion et Difference sont possibles pour les ensembles, les sélections temporaires ne peuvent être combinées avec d'autres sélections temporaires.

Les similitudes entre les sélections temporaires et les ensembles sont les suivantes :

• Comme un ensemble, une sélection temporaire existe en mémoire.
• Une sélection temporaire et un ensemble stockent des références aux enregistrements. Si des enregistrements sont modifiés ou détruits, la sélection temporaire ou l'ensemble peuvent n'être plus valides.
• Comme un ensemble, une sélection temporaire repère l'enregistrement courant au moment où elle est créée.

Référence

FIXER DESTINATION RECHERCHE, Nommer les objets du langage 4D.

COPIER SELECTION ({table; }tempo)

Paramètre	Type		Description
table	Table	→	Table de laquelle il faut copier la sélection ou Table par défaut si ce paramètre est omis
tempo	Alpha	→	Nom de la sélection temporaire à créer

Description

COPIER SELECTION copie la sélection courante de table dans une sélection temporaire tempo. La table par défaut du process courant est utilisée si le paramètre optionnel table n'est pas spécifié. La sélection temporaire tempo contient une copie de la sélection. La sélection courante et l'enregistrement courant de table pour le process courant ne sont pas modifiés.

Une sélection temporaire ne contient pas les enregistrements, mais une liste triée des références aux enregistrements. Chaque référence à un enregistrement prend 4 octets en mémoire. Ceci signifie que lorsqu'une sélection est copiée à l'aide de la commande COPIER SELECTION, la mémoire requise est 4 octets multipliés par le nombre d'enregistrements dans la sélection. Comme les sélections temporaires restent en mémoire, il vous faut assez de mémoire pour la sélection temporaire ainsi que la sélection courante de la table pour le process.

4D Server : La sélection temporaire tempo ainsi que la sélection courante sont logées dans la mémoire du poste serveur. En conséquence, assurez-vous que le serveur dispose de suffisamment de mémoire.

Utilisez la commande EFFACER SELECTION pour libérer la mémoire utilisée par tempo.

Exemple

L'exemple suivant permet de vérifier s'il y a des factures impayées dans la table [Personnes]. La sélection est triée puis sauvegardée. Nous cherchons toutes les factures qui n'ont pas été payées. Ensuite, nous réutilisons la sélection et effaçons la sélection temporaire en mémoire :

```
TOUT SELECTIONNER([Personnes])
    `Permettre à l'utilisateur de trier la sélection
TRIER([Personnes])
    ` Stocker la sélection dans une sélection temporaire
                COPIER SELECTION([Personnes];"TriéeUtilisateur")
```

```
    ` Rechercher les factures impayées
CHERCHER([Personnes];[Personnes]FactureDue=Vrai)
    ` Si un enregistrement a été trouvé
Si (Enregistrements trouves([Personnes])>0)
        ` Informer l'utilisateur
    ALERTE("Oui, quelques factures n'ont pas été réglées.")
Fin de si
    ` Réutiliser la sélection temporaire triée
UTILISER SELECTION("TriéeUtilisateur")
    ` Effacer la sélection de la mémoire
EFFACER SELECTION("TriéeUtilisateur")
```

Référence

DEPLACER SELECTION, EFFACER SELECTION, Nommer les objets du langage 4D, UTILISER SELECTION.

CREER SELECTION SUR TABLEAU

version 6.7 (Modifiée)

CREER SELECTION SUR TABLEAU (table; tabEnrg{; tempo})

Paramètre	Type		Description
table	Table	→	Table de la sélection
tabEnrg	Tab Entier long | Booléen	→	Tableau de n° d'enregistrements, ou Tableau de booléens (Vrai = l'enregistrement est dans la sélection, Faux = il n'est pas dans la sélection)
tempo	Alpha	→	Nom de la sélection temporaire à créer, ou Appliquer la commande à la sélection courante si ce paramètre est omis ou vide

Description

La commande CREER SELECTION SUR TABLEAU construit la sélection temporaire tempo à partir :
• soit du tableau de numéros d'enregistrements absolus tabEnrg de la table table,
• soit du tableau de booléens tabEnrg ; dans ce cas, les valeurs du tableau indiquent l'appartenance (Vrai) ou non (Faux) de chaque enregistrement de table à la sélection tempo.

Si vous ne passez pas le paramètre tempo ou si vous passez une chaîne vide, la commande s'appliquera à la sélection courante de table, qui sera donc mise à jour.

Lorsque vous utilisez la commande avec un tableau d'entiers longs, tous les numéros du tableau représentent la liste des numéros d'enregistrements qui feront partie de la sélection tempo. Si un numéro est invalide (enregistrement non créé), l'erreur -10503 est générée.

Note : Attention, vous devez veiller à ce que le tableau ne contienne pas d'éléments ayant la même valeur, sinon la sélection résultante sera incorrecte.

Lorsque vous utilisez la commande avec un tableau de booléens, le Nième élément du tableau représente l'intégration (Vrai) ou non (Faux) de l'enregistrement numéro N dans la sélection tempo. En principe, le nombre d'éléments du tableau doit être égal au nombre d'enregistrements de table. Si le tableau est plus petit que le nombre d'enregistrements, seuls les enregistrements définis par le tableau pourront faire partie de la sélection.

Note : Avec un tableau de booléens, la commande utilise les éléments du numéro 0 au numéro N-1.

Référence

COPIER SELECTION, CREER ENSEMBLE SUR TABLEAU, EFFACER SELECTION, Nommer les objets du langage 4D, TABLEAU ENTIER LONG SUR SELECTION, UTILISER SELECTION.

Gestion des erreurs

Si un numéro d'enregistrement est invalide (enregistrement non créé), l'erreur -10503 est générée. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

DEPLACER SELECTION ({table; }tempo)

Paramètre	Type		Description
table	Table	→	Table de la sélection ou Table par défaut si ce paramètre est omis
tempo	Alpha	→	Nom de la sélection temporaire à créer

Description

DEPLACER SELECTION crée la sélection temporaire tempo et y place la sélection courante de table. A la différence de COPIER SELECTION, cette commande ne copie pas la sélection, mais la déplace.

Après l'exécution de cette commande, la sélection courante de table dans le process courant est vide. En conséquence, DEPLACER SELECTION ne doit pas être utilisée lorsqu'un enregistrement est en cours de modification.

En termes d'utilisation de la mémoire, DEPLACER SELECTION est plus économique que COPIER SELECTION. En effet, COPIER SELECTION utilise 4 octets de mémoire pour chaque enregistrement de la sélection. Avec DEPLACER SELECTION, seule la référence à la sélection est déplacée.

Exemple

La méthode suivante vide la sélection courante de la table [Clients] :

```
DEPLACER SELECTION([Clients]; "AEffacer")
EFFACER SELECTION("AEffacer")
```

Référence

COPIER SELECTION, EFFACER SELECTION, Nommer les objets du langage 4D, UTILISER SELECTION.

EFFACER SELECTION (tempo)

Paramètre	Type		Description
tempo	Alpha	→	Nom de la sélection temporaire à effacer

Description

EFFACER SELECTION efface tempo de la mémoire et donc libère la mémoire qu'elle utilisait. EFFACER SELECTION n'affecte pas les tables, sélections courantes ou enregistrements. Comme les sélections temporaires utilisent de la mémoire, il est conseillé de les effacer si vous n'en avez plus besoin.

Si tempo a été créée par la commande DEPLACER SELECTION puis traitée à l'aide de la commande UTILISER SELECTION, elle n'existe plus en mémoire. Dans ce cas, vous n'avez pas besoin d'utiliser EFFACER SELECTION.

Référence

COPIER SELECTION, DEPLACER SELECTION, UTILISER SELECTION.

UTILISER SELECTION (tempo)

Paramètre	Type		Description
tempo	Alpha	→	Nom de la sélection temporaire à utiliser

Description

UTILISER SELECTION désigne la sélection temporaire tempo comme sélection courante pour la table à laquelle elle appartient.

Lorsque vous créez une sélection temporaire, l'enregistrement courant est aussi stocké par la sélection temporaire. UTILISER SELECTION récupère la position de cet enregistrement et en fait l'enregistrement courant. L'enregistrement courant est alors chargé. S'il a été modifié après la création de la sélection temporaire tempo, il doit être sauvegardé avant que la commande UTILISER SELECTION soit appelée, afin de ne pas perdre les informations modifiées.

• Si tempo a été créée par la commande COPIER SELECTION, la sélection temporaire est utilisée comme sélection courante de la table à laquelle elle appartient. La sélection temporaire tempo existe en mémoire jusqu'à ce qu'elle soit effacée. Pour récupérer l'espace mémoire occupé par tempo, appelez la commande EFFACER SELECTION.

• Si tempo a été créée par la commande DEPLACER SELECTION, elle est utilisée comme sélection courante de la table à laquelle elle appartient et tempo n'existe plus en mémoire.

N'oubliez pas qu'une sélection temporaire est la représentation d'une sélection courante à un instant donné. Si les enregistrements que la sélection temporaire représente sont modifiés, celle-ci devient obsolète. En conséquence, une sélection temporaire doit représenter une sélection d'enregistrements dont le contenu est relativement stable.
Différents événements peuvent rendre une sélection temporaire obsolète : la modification ou la suppression d'un enregistrement appartenant à la sélection temporaire ou la modification des critères de création de la sélection temporaire.

Référence

COPIER SELECTION, DEPLACER SELECTION, UTILISER SELECTION.

54

Serveur Web

4D en mode local, 4D en mode distant et 4D Server contiennent un serveur Web qui vous permet de publier des bases 4D ou tout type de page HTML sur le Web. Les principales caractéristiques du moteur du serveur Web de 4D sont les suivantes :

• Simplicité de publication
Vous pouvez à tout moment lancer ou stopper la publication de la base sur le Web. Pour cela, il suffit de choisir une commande de menu ou d'exécuter une commande du langage.

• Mode contextuel et Mode sans contexte
Le serveur Web de 4D peut fonctionner dans deux modes distincts : le mode contextuel et le mode sans contexte. Vous pouvez utiliser le serveur Web 4D dans le mode que vous souhaitez : mode contextuel, mode sans contexte, et passer à la volée d'un mode à l'autre en fonction de vos besoins.
- Le **mode contextuel** (disponible avec le serveur Web de 4D en mode local et de 4D Server uniquement) constitue une fonctionnalité unique et inégalée. Dans ce mode, 4D gère les navigateurs Web comme des clients standard de la base. Votre base est directement publiée sur le Web. Vous n'avez pas besoin de développer une base de données, un site Web et ensuite une interface CGI entre les deux. Votre base de données est votre site Web. Toute modification effectuée sur la structure ou les données de la base est immédiatement répercutée sur tous les navigateurs qui s'y connectent. 4D convertit à la volée en HTML les barres de menus, formulaires et méthodes de votre base : il n'est pas nécessaire de connaître le HTML pour publier une base 4D sur le Web. 4D maintient automatiquement un contexte d'utilisation des données pour chaque navigateur Web (sélections, variables, etc.). A noter qu'en contrepartie, la navigation Web en mode contextuel inclut des contraintes spécifiques. Pour plus d'informations, reportez-vous à la section Utiliser le Mode contextuel.
- Utilisé en **mode sans contexte** (mode standard), le serveur Web 4D est un serveur HTTP parfaitement standard : les pages Web sont envoyées sans qu'il soit nécessaire de maintenir de contexte. Vous pouvez accéder aux données de la base 4D et construire à la volée des pages HTML "semi-dynamiques" comportant des données statiques et des données issues de la base, avant de les envoyer aux navigateurs Web. Vous pouvez également envoyer des pages Web statiques ne nécessitant aucun traitement de la part du serveur Web.

• Méthodes base dédiées
La Méthode base Sur authentification Web et la Méthode base Sur connexion Web constituent les points d'entrée des requêtes dans le serveur Web ; elles peuvent être utilisées pour évaluer et acheminer tout type de requête.

• Utilisation de balises et d'URLs spéciaux
Le serveur Web de 4D propose de nombreux mécanismes permettant d'interagir avec les actions des utilisateurs, notamment :
- des balises spéciales peuvent être incluses dans les pages Web afin de provoquer des traitements par le serveur Web au moment de leur envoi aux navigateurs.

- des URLs spéciaux permettent d'appeler 4D afin d'exécuter toute action.
- ces URLs peuvent également être utilisés comme actions de formulaire pour déclencher des traitements lorsque l'utilisateur poste des formulaires HTML.

• Sécurité des accès
Des options de configuration automatiques vous permettent d'accorder des autorisations d'accès spécifiques aux navigateurs Web ou d'utiliser le système de mots de passe intégré de 4D. Vous pouvez définir un “Utilisateur Web générique” pour simplifier la gestion des accès à l’intérieur de la base.
La Méthode base Sur authentification Web vous permet d'évaluer toute requête avant qu'elle ne soit traitée par le serveur Web. La définition d’un dossier racine HTML par défaut vous permet de verrouiller les accès aux fichiers sur le disque.
Enfin, vous devez désigner individuellement les méthodes projet pouvant être exécutées via le Web.

• Connexions SSL
Le serveur Web 4D peut communiquer en mode sécurisé avec les navigateurs Web, à l'aide du protocole SSL (Secured Socket Layer). Ce protocole, compatible avec la majorité des navigateurs Web, permet d'authentifier les intervenants et garantit la confidentialité et l'intégrité de l'information échangée.

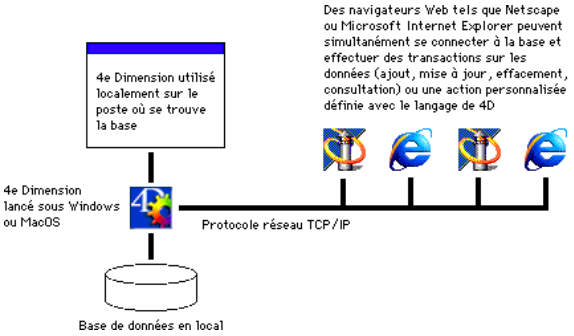
• Support étendu des formats Internet
Le serveur Web 4D est compatible HTTP/1.1, il peut gérer des documents XML et supporte la technologie WML (Wireless Markup Language).

• Support des CGI
Le serveur Web 4D peut très simplement utiliser des CGI, et peut également être interrogé par d'autres serveurs HTTP via des CGI.

• Exploitation simultanée des bases de données

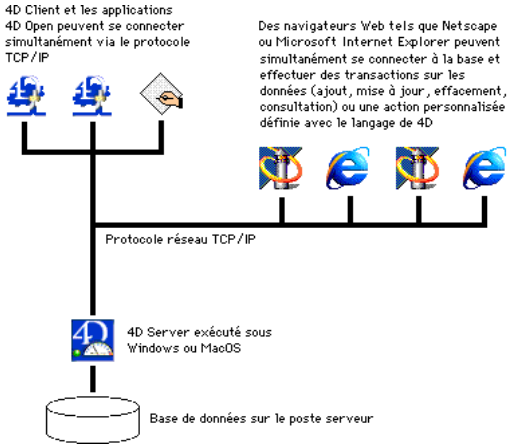
4D en mode local et le Web

Lorsqu'une base 4D est publiée sur le Web avec 4D en mode local, il est possible, simultanément :
- d'exploiter la base localement avec 4D
- de se connecter à la base avec un navigateur Web



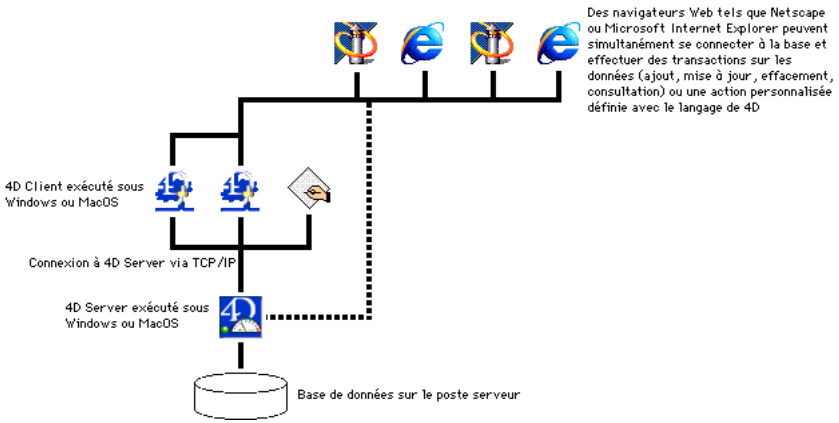
4D Server et le Web

Lorsqu'une base 4D est publiée sur le Web avec 4D Server, il est possible de se connecter à la base 4D et de l'utiliser simultanément :
- à partir de postes 4D distants
- à partir d'applications exploitant 4D Open
- à partir de navigateurs Web



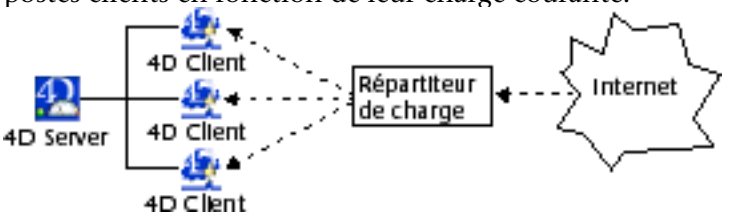
4D Client et le Web

Lorsqu'une base 4D est publiée sur le Web avec un client 4D, il est possible de se connecter à la base 4D et de l'utiliser simultanément :
- à partir de postes 4D distants
- à partir d'applications exploitant 4D Open
- à partir de navigateurs Web. Dans ce cas, si la base est également publiée avec 4D Server, les navigateurs Web peuvent se connecter à la base publiée via un poste 4D client ou via 4D Server. Ce fonctionnement permet notamment de gérer des modes d'accès différents aux données (public, administration, etc.).



Les mécanismes élémentaires du serveur Web de 4D sont exploités de manière semblable par 4D en mode distant, à l'exception du mode contextuel. En effet, il n'est pas possible d'utiliser le mode contextuel avec le serveur Web de 4D en mode distant (pour plus d'informations sur ce mode, reportez-vous à la section Utiliser le mode contextuel).
De même, le fonctionnement des commandes de langage est généralement identique, que la commande soit exécutée sur 4D en mode local, 4D Server ou 4D en mode distant. Le principe est que les commandes sont appliquées au site Web du poste sur lequel elles sont exécutées. Vous devez gérer ce principe à l'aide des commandes Exécuter sur serveur / Exécuter sur Client.

• **Répartition de charge avec les clients 4D** : tout poste 4D en mode distant pouvant être utilisé comme serveur Web, vous pouvez mettre en place un système de serveur Web dynamique avec répartiteur de charge. Les possibilités offertes sont vastes, notamment :
- la mise en place d'un système de répartition de charge (load balancing) afin d'optimiser les performances du serveur Web 4D : une réplique du site Web étant installée sur chaque serveur Web de 4D, un répartiteur de charge (matériel ou logiciel) adressera les requêtes aux postes clients en fonction de leur charge courante.



- la mise en place d'un serveur Web à "tolérance de panne" : le site Web 4D est répliqué sur deux ou plusieurs postes clients 4D. En cas de défaillance d'un serveur Web 4D, un autre prend le relais.
- la création de vues différentes des mêmes données, par exemple en fonction de la provenance des requêtes. Dans le cadre d'un réseau d'entreprise, un serveur Web de poste client 4D protégé peut servir les requêtes Intranet et un autre serveur Web de poste client 4D, situé au-delà du firewall, sert les requêtes Internet.
- la répartition des tâches entre les différents serveurs Web des clients 4D : un serveur Web de 4D peut être chargé des requêtes SOAP, un autre des requêtes standard, etc.

Référence

ARRETER SERVEUR WEB, ENVOYER FICHIER HTML, FIXER LIMITES AFFICHAGE WEB, FIXER PAGE ACCUEIL, FIXER RACINE HTML, FIXER TEMPORISATION WEB, Paramétrages du serveur Web, Sécurité des connexions, Support des CGI, Utiliser le mode contextuel, Utiliser le protocole SSL.

4D et 4D Server contiennent un serveur Web vous permettant de publier de manière transparente et dynamique les données de vos bases sur le Web.
Cette section présente les étapes nécessaires à la publication des bases 4D et à la connexion de navigateurs, ainsi que les process de gestion des connexions.

Conditions de publication d'une base 4D sur le Web

Pour pouvoir publier une base 4D sur le Web à l'aide de 4D ou de 4D Server, vous devez disposer des éléments décrits ci-dessous :

• une licence "4D Web Application". Pour plus d'informations sur ce point, reportez-vous au guide d'installation de 4D.

• les connexions Web sont effectuées par le biais du protocole réseau TCP/IP. Par conséquent :

- le protocole TCP/IP doit être installé et correctement configuré sur votre machine. Reportez-vous à la documentation de votre ordinateur ou de votre système d'exploitation pour plus d'informations sur ce point.

- Si vous voulez utiliser le protocole SSL pour vos connexions, assurez-vous que les composants nécessaires sont correctement installés (reportez-vous à la section Utiliser le protocole SSL).

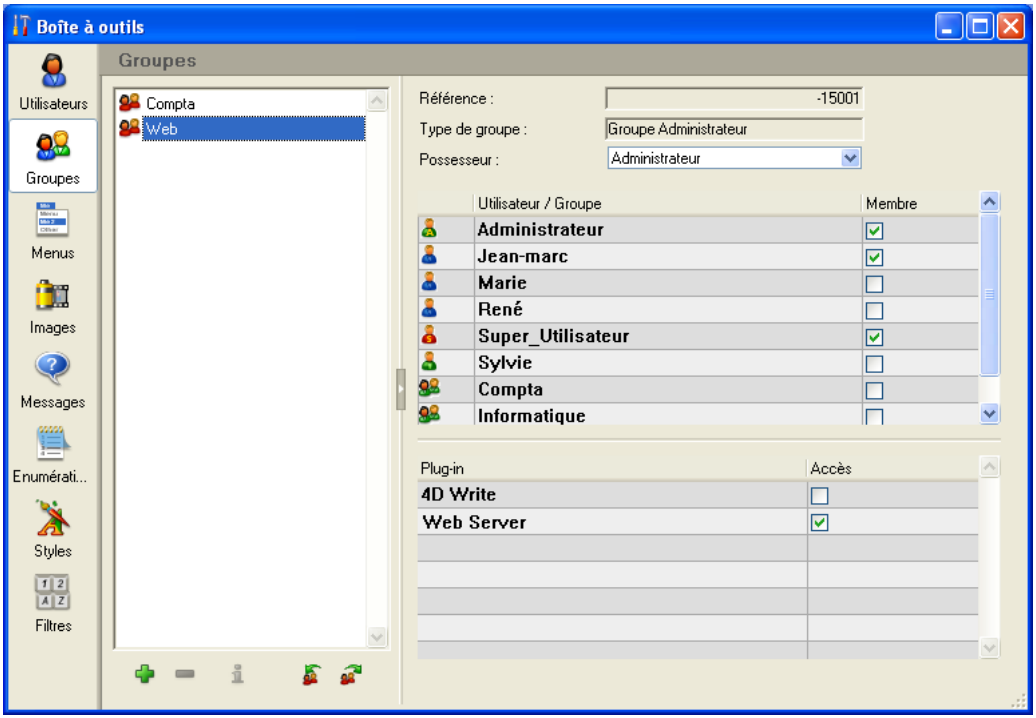
• Une fois les points précédents contrôlés et réglés, vous devez démarrer le serveur Web depuis 4D. Ce point est traité plus loin dans cette section.

Autorisation de publication (4D en mode distant)

Par défaut, tout poste client 4D peut publier sur le Web la base à laquelle il est connecté. Vous pouvez toutefois contrôler la possibilité de publication Web de chaque poste 4D distant en utilisant le système de mots de passe de 4D.
En effet, les licences Web 4D sont considérées par 4D Server comme des licences de plug-ins. Ainsi, comme pour un plug-in, vous pouvez restreindre le droit d'utiliser les licences Web Server à un groupe d'utilisateurs spécifique.

Pour cela, affichez la page **Groupes** dans la Boîte à outils depuis 4D (vous devez disposer des autorisations d'accès adéquates pour modifier ces paramètres).

Sélectionnez un groupe dans la liste de gauche, puis cochez l'option **Accès** en regard de la ligne **Web Server** dans la zone de répartition des plug-ins :

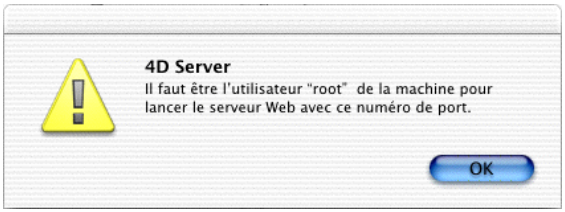


Ci-dessus : seuls les utilisateurs appartenant au groupe “Web” sont autorisés à publier leur 4D en tant que serveur Web.

Configuration spécifique sous Mac OS X

Sous Mac OS X, l’utilisation des ports TCP/IP réservés à la publication Web requiert des privilèges d’accès spécifiques : seul l’utilisateur “root” de la machine peut lancer une application utilisant ces ports.
Il s’agit des ports 0 à 1023. Par défaut, une base 4D est publiée sur le port TCP 80 en mode standard et 443 en mode SSL.

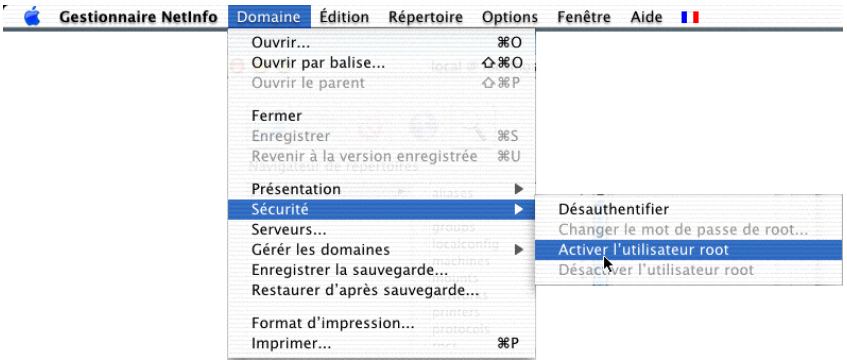
Lorsque vous publiez une base 4D sur le port TCP par défaut sans vous être connecté en tant qu'utilisateur "root", une boîte de dialogue d'alerte vous le signale :



Pour pouvez utiliser le serveur Web sous Mac OS X, vous disposez de quatre possibilités :

• **Modifier les numéros des ports TCP utilisés par le serveur Web 4D**.
Vous devez utiliser des numéros de ports supérieurs à 1023, par exemple 8080 pour le mode standard et 8043 pour le mode SSL.
Cette opération s'effectue dans la boîte de dialogue des Préferences de la base (cf. section Paramétrages du serveur Web) ou à l'aide de la commande FIXER PARAMETRE BASE. Dans ce cas, il sera nécessaire d'indiquer le numéro de port derrière chaque URL de connexion à la base (par exemple, http://www.mabase.fr/pages/mapage.html:8080 et https://www.mabase.fr/pages/paiement.html:8043.

• **Se connecter en tant qu'utilisateur "root"**
Par défaut, l'utilisateur "root" n'est pas actif sur un poste sous Mac OS X. Vous devez d'abord l'activer, puis vous connecter sous ce nom d'utilisateur.
L'activation de l'utilisateur "root" s'effectue par l'intermédiaire de l'utilitaire **Gestionnaire NetInfo** (ou NetInfo Manager), fourni par Apple et installé dans le dossier Applications:Utilities.
Une fois l'utilitaire lancé, choisissez la commande **Sécurité** dans le menu **Domaine**, puis l'option **Activer l'utilisateur root**. Vous devez auparavant avoir identifié un administrateur de la machine via la commande **Authentifier...**, située dans le même menu (saisissez le nom abrégé et le mot de passe d'un administrateur de la machine).



Pour plus d'informations sur cette opération, veuillez vous reporter à la documentation de Mac OS X. Une fois l'utilisateur "root" créé, vous devez fermer la session (menu Pomme) puis vous connecter sous le nom d'utilisateur "root". Il vous est alors possible de lancer le serveur Web sur le numéro de port 80, ou un serveur Web 4D en mode sécurisé.

• Transférer le port
Cette troisième solution permet de publier une base Web 4D sous Mac OS X sans qu'il soit nécessaire de spécifier le numéro de port derrière chaque URL de connexion au serveur et sans être l'utilisateur "root". Elle est basée sur le transfert de port. Le principe consiste à transférer, au niveau du système, les requêtes reçues sur le numéro de port TCP standard (80) à celui défini dans la base 4D (devra être supérieur à 1023). Notez que cette astuce ne fonctionne pas avec les connexions sécurisées (le port TCP 443 n'est pas modifiable).
Pour effectuer l'opération, vous devrez vous connecter en tant qu'utilisateur "root", lancer le Terminal et utiliser des commandes Unix.

Pour mettre en place le transfert de port sous Mac OS X (en supposant que votre adresse IP est 192.168.93.45) :
1. Ouvrez une session en tant qu'utilisateur root (cf. paragraphe précédent).
2. Lancez le programme **Terminal**.
Ce programme se trouve dans le dossier Applications:Utilities.
3. Saisissez "su" (compte spécial "utilisateur substitut") puis le mot de passe de l'utilisateur root.
4. Saisissez la commande suivante :
ipfw add 400 fwd 192.168.93.45,8080 tcp from any to 192.168.93.45 80
Bien entendu, vous devez remplacer "192.168.93.45" par votre propre adresse IP.
Le chiffre 400 est le numéro de référence de cette opération.
5. Quittez le programme **Terminal**.
6. Lancez votre application 4D en tant qu'utilisateur standard.
7. Dans les Préférences de la base, fixez le port TCP de publication Web à 8080.
Dès lors, Mac OSX est prêt à transférer instantanément les requêtes reçues sur le port 80 vers le port 8080, de manière transparente pour l'utilisateur.

Pour supprimer ce mode de fonctionnement :
1. Lancez le programme **Terminal** et saisissez :
ipfw delete 400
Les requêtes reçues sur le port 80 ne sont alors plus transférées au port 8080.

• Ouvrir le port via une application spécialisée
Le principe de cette solution consiste à déléguer l'ouverture du port Web à une application spécialisée, nommée HelperTool, disposant des privilèges suffisants. Ce mécanisme fonctionne avec 4D (tous modes), 4D Server et les applications exécutables 4D Volume Desktop.
L'application HelperTool est incluse dans le progiciel 4D. Elle doit être située à un emplacement spécifique du système. L'installation s'effectue automatiquement lors de la

première ouverture d'un port de numéro <1024 sur le poste. L'utilisateur est informé qu'un outil va être installé et est invité à saisir un nom et un mot de passe d'administrateur de la machine. Cette opération n'a lieu qu'une seule fois.
L'application est renommée "com.4D.HelperTool" et est installée dans le dossier "/Library/PrivilegedHelperTools/". Après la séquence initiale, le serveur Web de 4D peut être démarré et stoppé de façon transparente.

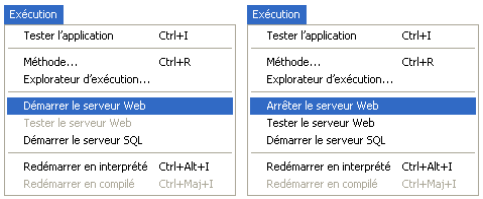
Note : Ce mécanisme nécessite au minimum la version 10.4.6 de Mac OS X. Si vous disposez d'une version antérieure du système, vous devez utiliser une autre solution de publication.

Démarrer le serveur Web 4D

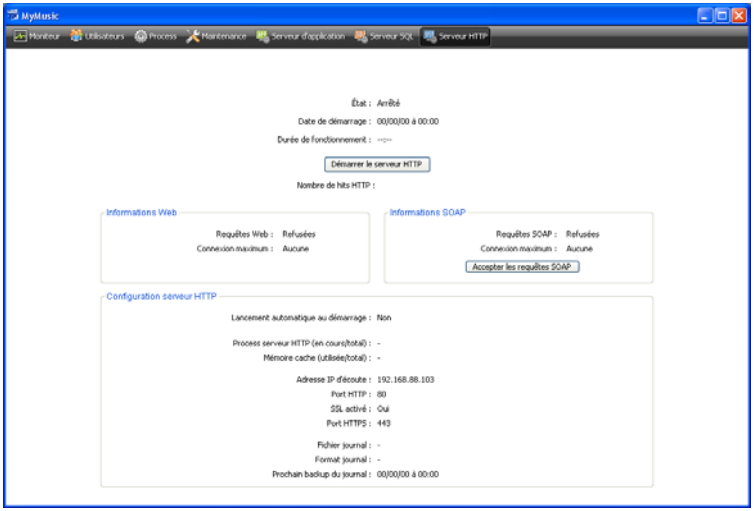
Le serveur Web 4D peut être démarré de trois manières différentes :

• par l'intermédiaire du menu **Exécuter** de 4D ou de la page Serveur HTTP de 4D Server (bouton **Démarrer le serveur HTTP**). Ces commandes vous permettent de lancer et d'arrêter le serveur Web à tout moment :

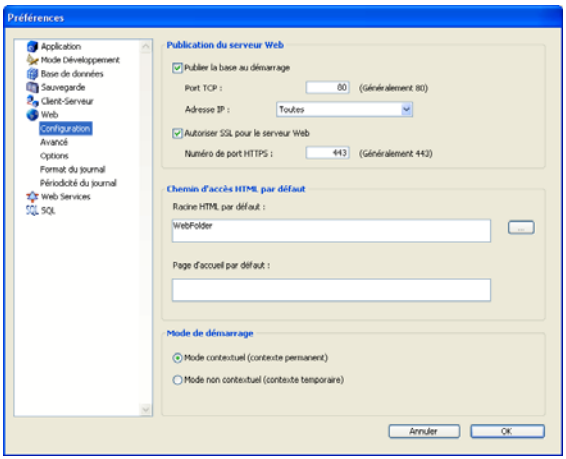
4D :



4D Server :



• par la publication automatique de la base à chaque fois qu'elle est ouverte. Pour que la base soit automatiquement publiée comme serveur Web, affichez la page **Configuration** du thème **Web** des Préférences de l'application :



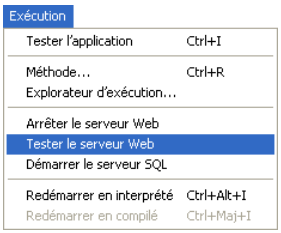
Dans la zone “Publication du serveur Web”, cochez la case **Publier la base au démarrage** puis cliquez sur le bouton **OK**. La base sera désormais automatiquement publiée comme serveur Web chaque fois que vous l'ouvrirez avec 4D ou 4D Server.

• par programmation, en appelant la commande LANCER SERVEUR WEB.

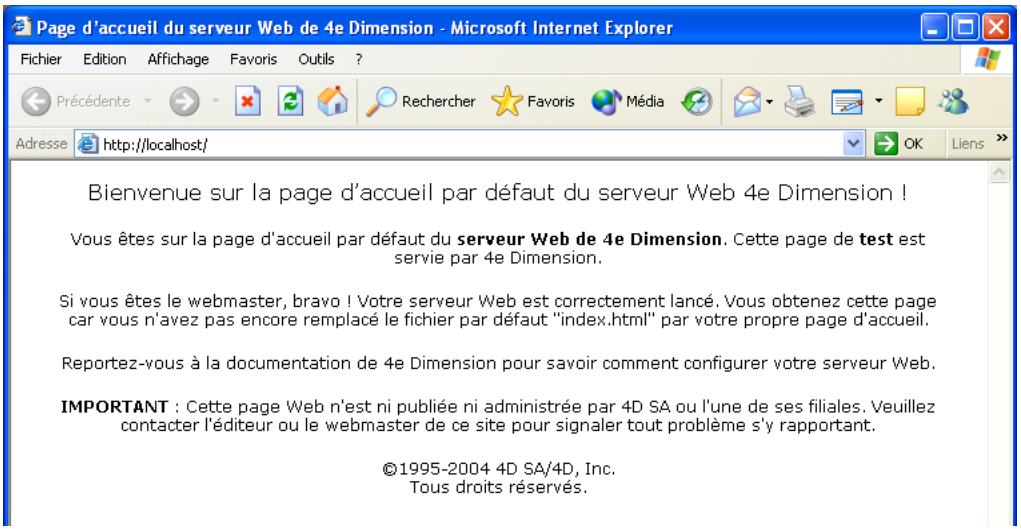
Note : Il n'est pas nécessaire de rouvrir votre base de données pour lancer ou arrêter sa publication comme serveur Web. Vous pouvez interrompre et redémarrer le serveur Web autant de fois que vous voulez à l'aide du menu **Exécution**, du bouton **Démarrer le serveur HTTP** ou en appelant les commandes LANCER SERVEUR WEB et ARRETER SERVEUR WEB.

Tester le serveur Web

La commande **Tester le serveur Web** permet de contrôler le fonctionnement du serveur Web intégré (4D uniquement). Cette commande est accessible dans le menu **Exécution** lorsque le serveur Web est lancé :



Lorsque vous sélectionnez cette commande, la page d'accueil du site Web publié par l'application 4D s'affiche dans une fenêtre de votre navigateur par défaut :



Cette commande permet de vérifier le fonctionnement du serveur Web, l'affichage de la page d'accueil, etc. La page est appelée via l'URL Localhost, qui est le raccourci standard désignant l'adresse IP de la machine sur laquelle est exécuté le navigateur. La commande tient compte du numéro de port TCP de publication spécifié dans les Préférences de l'application.

Connexion à une base 4D publiée sur le Web

Une fois que vous avez lancé la publication d'une base 4D sur le Web, vous pouvez vous y connecter avec un navigateur Web. Pour cela :

• Si votre site Web dispose d'un nom d'hôte enregistré (par exemple, "www.bellesfleurs.com"), il vous suffit d'indiquer ce nom dans la zone "Ouvrir", "Adresse" ou encore "Lieu" du navigateur puis d'appuyer sur la touche **Entrée** pour vous connecter.

• Si votre site Web ne dispose pas d'un nom enregistré, indiquez l'adresse IP de la machine de la base (par exemple 123.4.567.89) dans la zone "Ouvrir", "Adresse" ou encore "Lieu" du navigateur puis appuyez sur la touche **Entrée**.

A cet instant, votre navigateur doit afficher la page d'accueil de votre site Web. Si vous avez publié une base en conservant les paramétrages standard, vous devez obtenir la page d'accueil par défaut du serveur Web de 4D. Cette page vous permet de tester la connexion et le fonctionnement du serveur.

Vous pouvez également rencontrer une des situations décrites ci-dessous.

Note : Si votre base est protégée par un système de contrôle d'accès, il se peut que vous ayez à saisir un nom et un mot de passe (pour plus d'informations, reportez-vous à la section Sécurité des connexions).

(1) La connexion échoue, vous obtenez un message du type "...le serveur n'accepte pas de connexions ou est occupé...". Dans ce cas, effectuez les contrôles suivants :
• Vérifiez que le nom du serveur ou l'adresse IP que vous avez saisi(e) est correct(e).
• Vérifiez que 4D ou 4D Server est bien lancé et que le serveur Web a bien démarré.
• Vérifiez que la base de données est bien configurée pour être publiée sur le port TCP Web par défaut, c'est-à-dire 80 (voir aussi point 4).
• Vérifiez que le protocole réseau TCP/IP est correctement configuré sur la machine serveur et sur la machine du navigateur (les deux machines doivent se trouver sur le même réseau ou sous-réseau, ou les routeurs doivent être correctement configurés).
• Vérifiez les connexions physiques.
• Si vous ne testez pas localement votre propre site mais essayez de vous connecter à une base Web publiée sur Internet ou Intranet par quelqu'un d'autre, il se peut qu'en définitive le message décrive une situation réelle : le poste serveur peut être éteint ou occupé, dans ce cas vous pouvez tenter de vous connecter ultérieurement ou contacter l'administrateur du site Web.

(2) La connexion est établie, mais vous obtenez une erreur HTTP 404, "Fichier non trouvé". Ce cas signifie que la page d'accueil du site n'a pu être servie. Dans ce cas, vérifiez que la page d'accueil existe bien à l'emplacement défini dans les Préférences de la base (cf. section Paramétrages du serveur Web) ou à l'aide de la commande FIXER PAGE ACCUEIL.

(3) La connexion est établie, mais vous obtenez une page Web avec le message "Barre de menu/Cette base de données ne peut être publiée sur le Web telle quelle, vous devez d'abord créer une barre de menus". Cela signifie que vous êtes bien connecté à la base publiée en mode contextuel, mais qu'aucune page d'accueil ni barre de menus n'est définie (en mode contextuel, 4D publie la barre de menus n°1 comme page d'accueil par défaut si aucune page HTML n'est spécifiée). Pour plus d'informations, reportez-vous à la section Premiers pas.

(4) La connexion est établie, mais vous n'obtenez pas la page Web que vous attendiez ! Cela peut se produire lorsque plusieurs serveurs Web sont exécutés simultanément sur la même machine. Par exemple :

• Vous avez lancé une seule base Web 4D, mais sur un système Windows qui exécute déjà son propre serveur Web.

• Vous avez lancé plusieurs bases Web 4D sur la même machine.

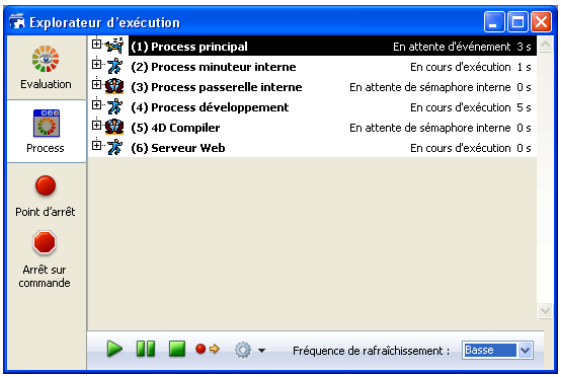
Dans les cas décrits ci-dessus, il vous suffit de changer les numéros des ports TCP sur lesquels vos bases 4D Web sont publiées. Pour cela, reportez-vous à la section Paramétrages du serveur Web.

Gestion des process Web

Divers process 4D prennent en charge la publication Web des bases de données et la connexion des navigateurs. Ce paragraphe décrit ces process ainsi que leurs caractéristiques.

Process Serveur Web

Le process Serveur Web s'exécute lorsque la base est publiée en tant que serveur Web. Dans la page **Process** de l'Explorateur d'exécution présentée ci-dessous, le process Server Web est le cinquième process :



Ce process est un process du noyau de 4D, vous ne pouvez donc pas l'arrêter à l'aide de la commande **Tuer**. De même, vous ne pouvez pas effectuer de communication interprocess à l'aide des commandes comme APPELER PROCESS. Notez que le process Serveur Web n'a pas d'éléments d'interface (fenêtres, menus, etc.).

Vous pouvez démarrer le process Serveur Web :
• en cliquant sur le bouton **Démarrer le serveur HTTP** dans la page "Serveur HTTP" de 4D Server ou en choisissant **Démarrer le serveur Web** dans le menu **Exécution** de 4D.
• en appelant la commande LANCER SERVEUR WEB.
• en ouvrant une base pour laquelle la préférence **Publier la base au démarrage** a été sélectionnée.

Vous pouvez arrêter le process Serveur Web :
• cliquant sur le bouton **Arrêter le serveur HTTP** dans la page "Serveur HTTP" de 4D Server ou en choisissant **Arrêter le serveur Web** dans le menu **Exécution** de 4D.
• en appelant la commande ARRETER SERVEUR WEB.
• en quittant une base publiée comme serveur Web.

Le rôle du process Serveur Web est de gérer les tentatives de connexion Web. Lorsque vous démarrez le process Serveur Web, vous n'ouvrez pas de connexion Web, vous permettez aux utilisateurs Web de se connecter à la base. Lorsque vous arrêtez le process Serveur Web, vous ne fermez pas les process de connexion Web ouverts (s'il y en a), simplement vous ne permettez plus à des utilisateurs Web de se connecter à la base.

Si des process de connexion Web étaient ouverts au moment où vous arrêtez le process Serveur Web, chacun de ces process continue à s'exécuter normalement.
Par conséquent, un délai d'attente peut être nécessaire avant l'arrêt complet du process Serveur Web.

Process Web et process Connexion Web
Chaque fois qu'un navigateur Web tente de se connecter à la base, la requête est gérée par le process Serveur Web, qui procède de la manière suivante :

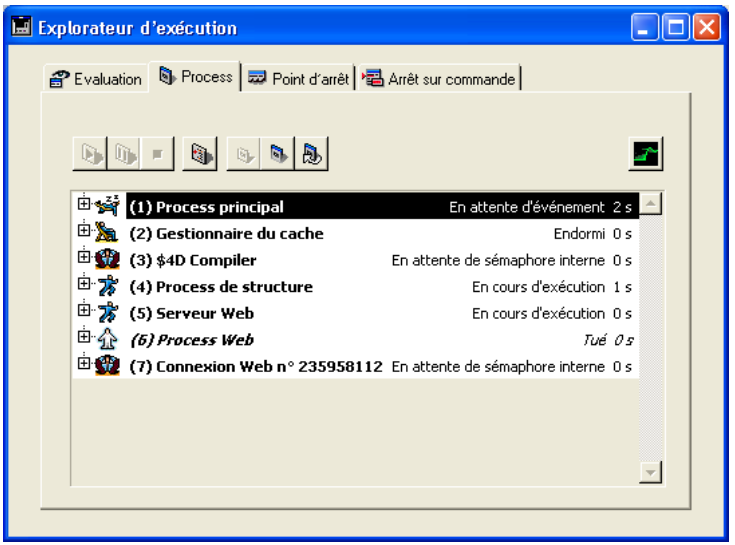
• D'abord, il crée un ou plusieurs process 4D locaux appelés **Process Web** pour gérer et évaluer la connexion au navigateur Web.

Note : Ces process gèrent toutes les requêtes HTTP. Ils s'exécutent très rapidement puis sont “tués” ou endormis. En effet, à des fins d'optimisation du serveur Web, une fois qu'il a traité une requête, un process de connexion Web temporaire est gelé pendant quelques secondes pour être éventuellement réactivé lorsqu'une autre requête arrive. Ce mécanisme peut être ajusté (délai d'attente, nombre minimum et maximum de process à conserver dans la "réserve" de process) à l'aide de la commande FIXER PARAMETRE BASE.

• Si la requête ne nécessite pas la création d'un contexte, le process Web prend en charge le traitement de la requête et l'envoi de la réponse éventuelle au navigateur. Le process temporaire est alors tué ou endormi (cf. ci-dessus).

• Si la requête nécessite la création d'un contexte, il vérifie alors si les ressources disponibles sont suffisantes. Si cela n'est pas le cas, il envoie au navigateur Web le message suivant : “Cette base de données n'a pas encore été paramétrée pour le Web”.
Si la connexion Web est correctement effectuée, un process **Connexion Web** est créé. Ce process gèrera toute la session Web pour cette connexion.

La liste des process, présentée ci-dessous, affiche le process de connexion Web “Connexion Web n° 235958112” démarré à la suite de la connexion du navigateur Web :



Notez que le sixième process, qui a été démarré puis tué, a géré l'initialisation de la connexion Web.

Note : Pour plus d'informations sur la gestion des contextes, reportez-vous à la section Utiliser le mode contextuel.

• Si, au cours de la session, la connexion passe du mode contextuel au mode sans contexte, le process de connexion Web (numéroté) est immédiatement tué. A l'inverse, si au cours de la session, la connexion passe du mode sans contexte au mode contextuel, un process de connexion Web numéroté est créé.

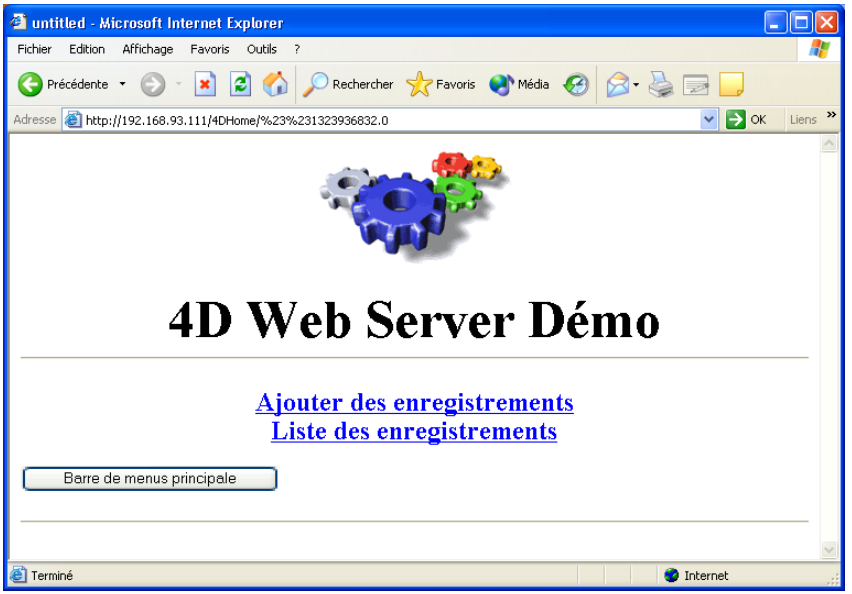
Référence

ARRETER SERVEUR WEB, ENVOYER FICHIER HTML, FIXER LIMITES AFFICHAGE WEB, FIXER RACINE HTML, FIXER TEMPORISATION WEB, Utiliser le protocole SSL.

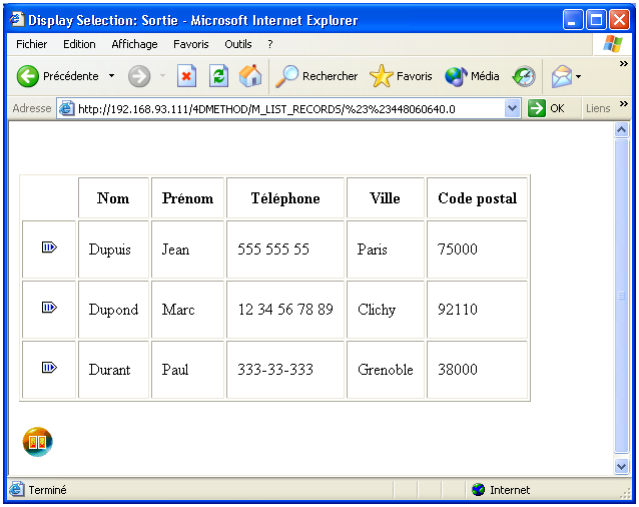
Exemple en mode contextuel

Cette section présente un exemple simple de publication instantanée d'une base de données en mode contextuel. Ce mode automatique peut notamment être utilisé pour les serveurs Intranet. Il permet d'illustrer les principes élémentaires de fonctionnement du serveur Web de 4D. Le fonctionnement et la structure de cette base sont classiques : la base comporte une table, un formulaire entrée, un formulaire sortie et une barre de menus. La page d'accueil est personnalisée.

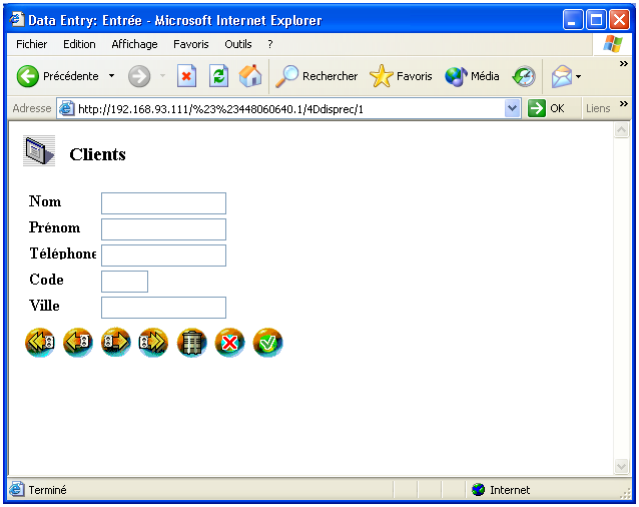
Lorsqu'un navigateur se connecte au serveur Web 4D, il obtient la page d'accueil suivante :



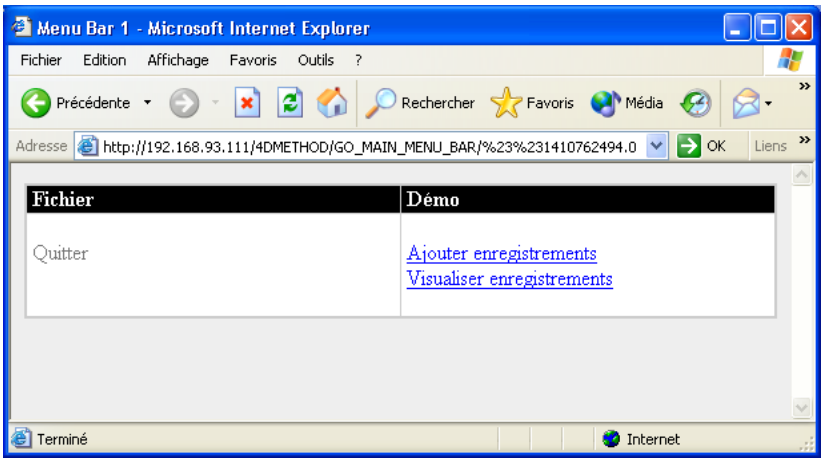
Si vous cliquez sur le lien 'Liste des enregistrements', vous obtenez l'équivalent de l'affichage d'une sélection 4D sur le Web :



Vous pouvez dès lors naviguer à votre convenance parmi les enregistrements. Lorsque vous cliquez sur le bouton 'Terminé', vous retournez à la page d'accueil du site Web.
Si vous cliquez sur le lien 'Ajouter des enregistrements' dans la page d'accueil, vous obtenez l'équivalent d'un ajout d'enregistrement 4D sur le Web :



Vous pouvez ajouter autant d'enregistrements que vous voulez. Lorsque vous cliquez sur le bouton 'Terminé' (celui avec le "X") vous retournez à nouveau à la page d'accueil du site Web. Si vous cliquez sur le bouton **Barre de menus principale** dans la page d'accueil, vous obtenez la barre de menus créée de 4D sur le Web :

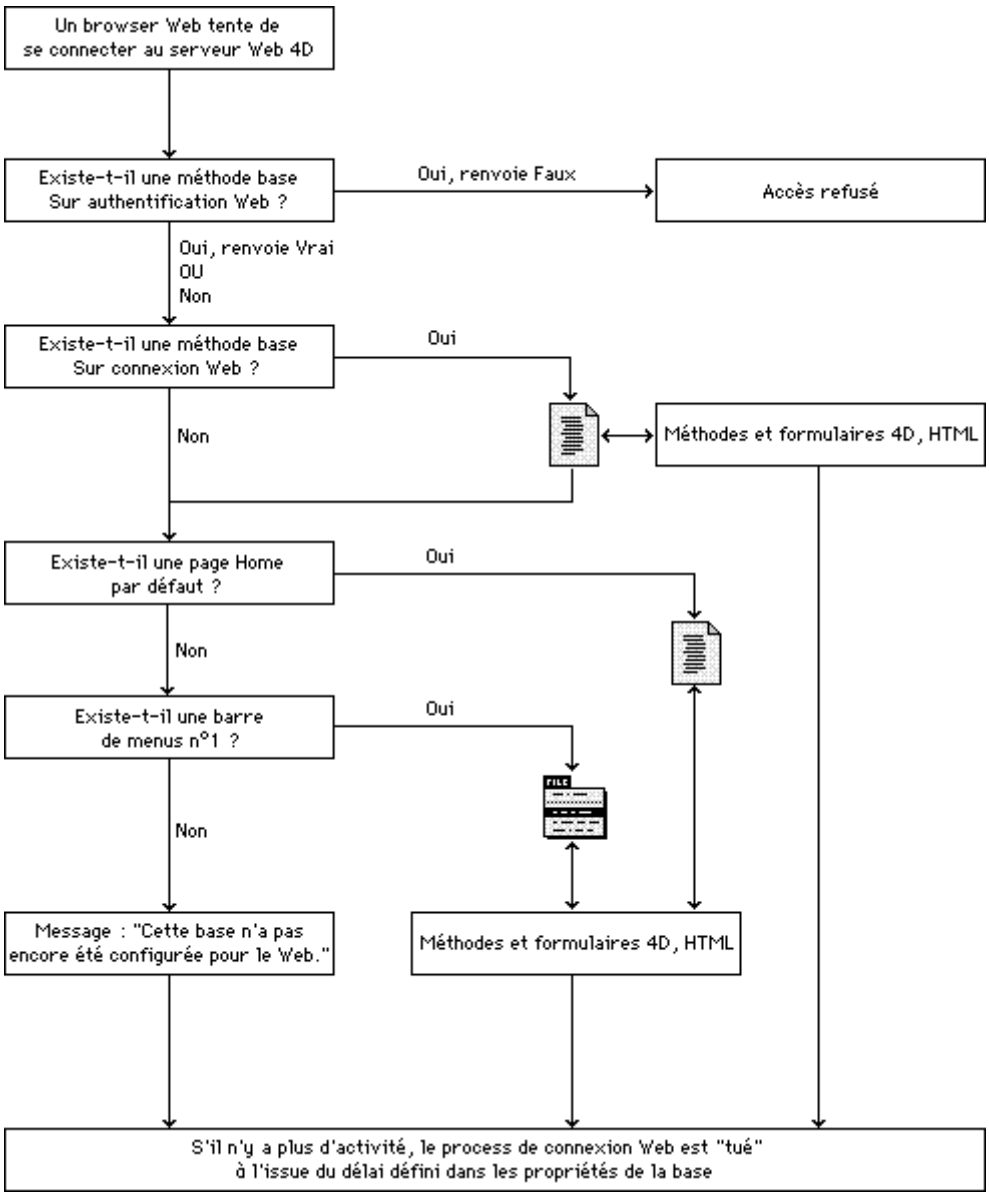


Vous pouvez alors cliquer sur une commande de menu pour visualiser ou ajouter des enregistrements. Les méthodes 4D associées à ces commandes de menu sont les mêmes que celles qui étaient utilisées à partir de la page d'accueil.
Lorsque vous avez terminé, vous quittez le navigateur. 4D terminera la connexion Web dès que le délai d'attente avant déconnexion se sera écoulé.

Démarrage d'une connexion Web en mode contextuel

Chaque fois qu'un navigateur Web se connecte à une base 4D publiée en tant que serveur Web en mode contextuel, 4D effectue par défaut les actions suivantes :
• La Méthode base Sur authentification Web est exécutée, si elle existe.
• Si cette méthode base n'existe pas ou si elle retourne Vrai, la Méthode base Sur connexion Web est exécutée, si elle existe.
• Si cette méthode base n'existe pas ou si son exécution est terminée, 4D affiche la page d'accueil par défaut définie dans les Préférences de la base, s'il y en a une.
• Si aucune page d'accueil par défaut n'est définie, 4D affiche la barre de menus courante (par défaut, la barre de menus n°1), si elle existe.
• S'il n'y a pas de page d'accueil par défaut ni de barre de menus définie dans la base, 4D affiche une page de Web qui indique :“Cette base n'a pas encore été configurée pour le Web”.

Cette séquence est résumée dans le schéma suivant :



La Méthode base Sur connexion Web peut appeler toute méthode ou tout formulaire défini(e) dans la base, ainsi que des pages HTML. En définitive, cette méthode base peut gérer la totalité de la session.

Une connexion Web à 4D ou 4D Server n'est pas de même nature qu'une connexion client/serveur. Le protocole HTTP, qui supporte HTML et le Web, n'est pas un protocole basé sur la session. C'est plutôt un protocole basé sur la requête. En client/serveur, vous vous connectez, effectuez une session de travail, et finalement vous vous déconnectez du serveur. Avec HTTP, chaque fois que vous réalisez une action qui nécessite l'attention ou une action du serveur Web, une requête est envoyée au serveur. Pour résumer, une requête HTTP peut être considérée comme une séquence connexion+requête+attente de réponse+déconnexion.

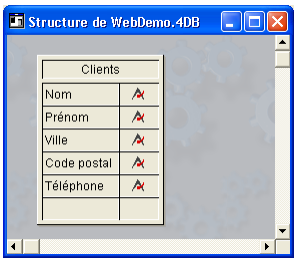
En mode contextuel, pour maintenir une session client/serveur réelle par l'intermédiaire de HTTP, par défaut 4D gère pour vous, via un encodage transparent des URLs, des contextes qui identifient de façon unique chaque connexion Web, et en même temps associent la connexion au process 4D qui la gère. Dans ce mode toutefois, 4D ne peut pas terminer une "session de requêtes" Web de la même manière qu'une session client/serveur standard. C'est pour cette raison que la clôture de ces sessions client/serveur est gérée par un système de temporisation (timeout). Le process 4D qui gère la connexion Web est tué lorsque la période d'inactivité maximale autorisée pour la base est atteinte.

Base de données et serveur Web à la fois

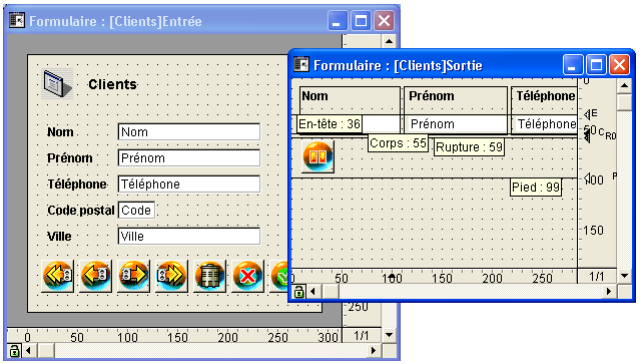
Une session de serveur Web 4D peut être entièrement gérée à l'aide des barres de menus, formulaires et méthodes 4D en mode contextuel. Dans l'exemple précédent, la visualisation et l'ajout des enregistrements étaient effectués par l'intermédiaire de méthodes et formulaires 4D simples. Si nous n'avions pas inclus de page d'accueil en HTML, la barre de menus n°1 aurait été directement affichée lors de la connexion Web.

Si nous mettons de côté la page HTML d'accueil, la construction d'un serveur Web qui supporte les transactions de la base client/serveur revient à développer une base 4D comme sous Windows ou Mac OS, en mono ou multi-utilisateur.

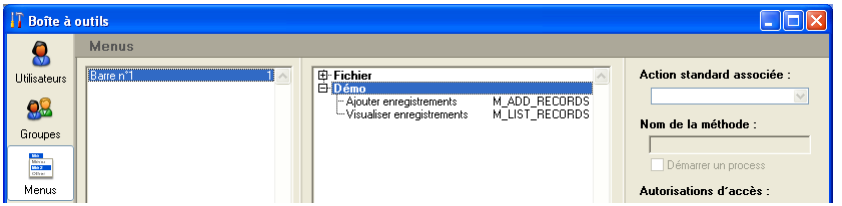
• Voici la structure de la base exemple :



• Pour visualiser les enregistrements, les formulaires entrée et sortie suivants ont été créés :



• Pour travailler avec les menus personnalisés et gérer les connexions Web, la barre de menus n°1 suivante a été créée :



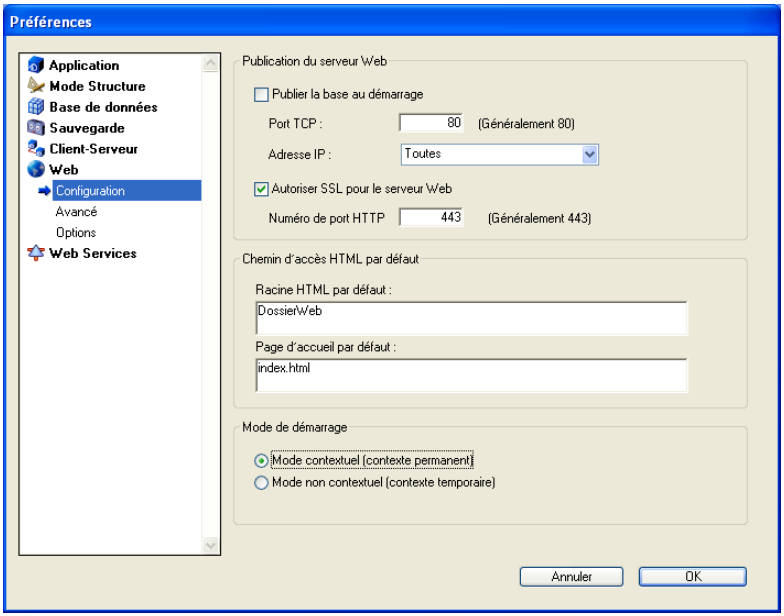
• Les deux méthodes projet suivantes ont été écrites :

```
      `Méthode projet M_ADD_RECORDS
C_TEXTE($1)  ` Ce paramètre DOIT être explicitement déclaré
Repeter
   AJOUTER ENREGISTREMENT ([Clients])
Jusque (OK=0)
```

```
    `Méthode projet M_LIST_RECORDS
C_TEXTE($1)  ` Ce paramètre DOIT être explicitement déclaré
TOUT SELECTIONNER ([Clients])
MODIFIER SELECTION([Clients])
```

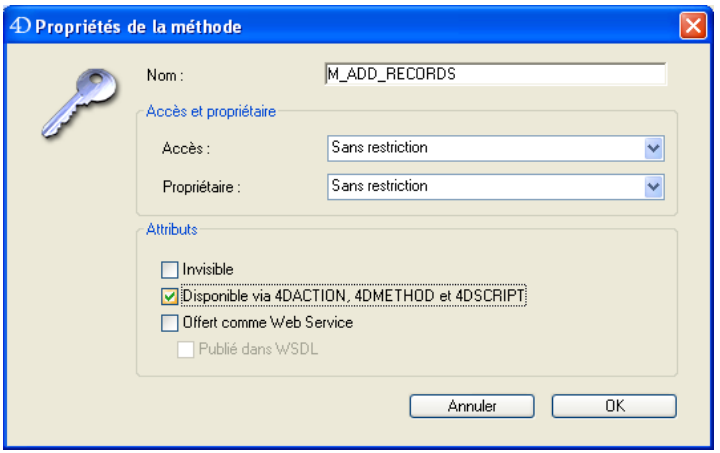
L'option “Démarrer un process” a été associée à chaque méthode dans l'éditeur de menus.

• Le serveur Web démarre en mode contextuel et une page d'accueil par défaut est définie dans les Préférences de la base :



• La page d’accueil contient deux liens, "Ajouter des enregistrements" et "Liste des enregistrements", qui déclenchent l'exécution des méthodes projet 4D M_ADD_RECORDS et M_LIST_RECORDS par l'intermédiaire de leurs URLs. La convention est simple : tout objet HTML comportant un lien peut appeler une méthode projet de votre base par l'URL "/4DMETHOD/Nom_de_votre_Méthode".

L'attribut **Disponible via 4DACTION, 4D METHOD et 4D SCRIPT** doit être associé à chaque méthode appelée via 4DMETHOD :



Une fois ces liens définis, 4D exécute la méthode projet spécifiée après le mot-clé /4DMETHOD/ lorsqu'un navigateur Web retourne l'URL. Ensuite, lorsque la méthode projet se termine, vous retournez à la page HTML qui a lancé son exécution. Notez que la méthode projet peut elle-même appeler des formulaires 4D, d'autres pages HTML, et ainsi de suite.
Votre site Web 4D peut être un système basé entièrement sur 4D ou un mélange de formulaires 4D et de pages HTML. Si vous utilisez des pages HTML dans votre base 4D, vous bénéficiez des deux environnements : 4D et HTML. Mais rappelez-vous que vous n'êtes absolument pas obligé d'utiliser des pages HTML, si vous le souhaitez !

• La page d'accueil HTML utilisée dans l'exemple comprend également un bouton. Il existe trois types de boutons HTML : **normal**, **submit** et **reset**. Un bouton "normal" peut se voir attribuer un URL qui fait référence à une méthode 4D à l'aide du mot-clé /4DMETHOD/. Les boutons "reset" ne sont guère utiles dans un développement 4D : ils effacent les valeurs éventuellement saisies par l'utilisateur dans un formulaire et n'envoient pas de requête au serveur. Les boutons "submit" retournent le formulaire au serveur Web avec les valeurs éventuellement saisies par l'utilisateur. Lors de l'intégration des pages HTML dans 4D, vous utiliserez généralement des boutons de type "normal" ou "submit". Les boutons de type "normal" servent à naviguer parmi les pages et les boutons de type "submit" servent à gérer la saisie de valeurs à partir de pages HTML lorsque vous ne souhaitez pas utiliser pour cela de formulaires 4D standard. Le code du bouton de la page d'accueil est le suivant : INPUT TYPE="SUBMIT" NAME="/4DMETHOD/GO_MAIN_MENUBAR"

Pour gérer l'envoi du formulaire HTML du côté du serveur Web 4D, vous devez spécifier l'action POST pour la méthode 4D qui sera exécutée par 4D dès réception du formulaire. Pour cela, il doit contenir la ligne FORM ACTION="/4DMETHOD/GO_MAIN_MENU_BAR" METHOD="POST"

• La méthode projet GO_MAIN_MENU_BAR est la suivante :

```
FIXER PAGE ACCUEIL("")
```

Cette méthode n'a qu'un objectif : ne plus afficher la page d'accueil par défaut, et donc envoyer la barre de menus courante. 4D affiche alors la barre de menus n°1 de votre base.

C'est tout !

En moins de cinq minutes, lorsque vous créez une base 4D, elle peut être utilisée localement ou comme serveur Web que vous pouvez publier sur votre réseau Intranet ou sur Internet.

Référence

ARRETER SERVEUR WEB, ENVOYER FICHIER HTML, FIXER LIMITES AFFICHAGE WEB, FIXER TEMPORISATION WEB, LANCER SERVEUR WEB.

Vous pouvez sécuriser les connexions à votre serveur Web 4D à l'aide des éléments suivants :

• Pour les accès Web, la combinaison du système de gestion des mots de passe (mode BASIC ou mode DIGEST) et de la Méthode base Sur authentification Web,
• La définition d'un "Utilisateur Web générique",
• La définition d'un dossier racine HTML par défaut,
• La définition de la propriété "Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT" pour chaque méthode projet de la base.

Note : La sécurité des informations transmises via la connexion est prise en charge par le protocole SSL. Pour plus d'informations, reportez-vous à la section Utiliser le protocole SSL.

Gestion des mots de passe pour les accès Web

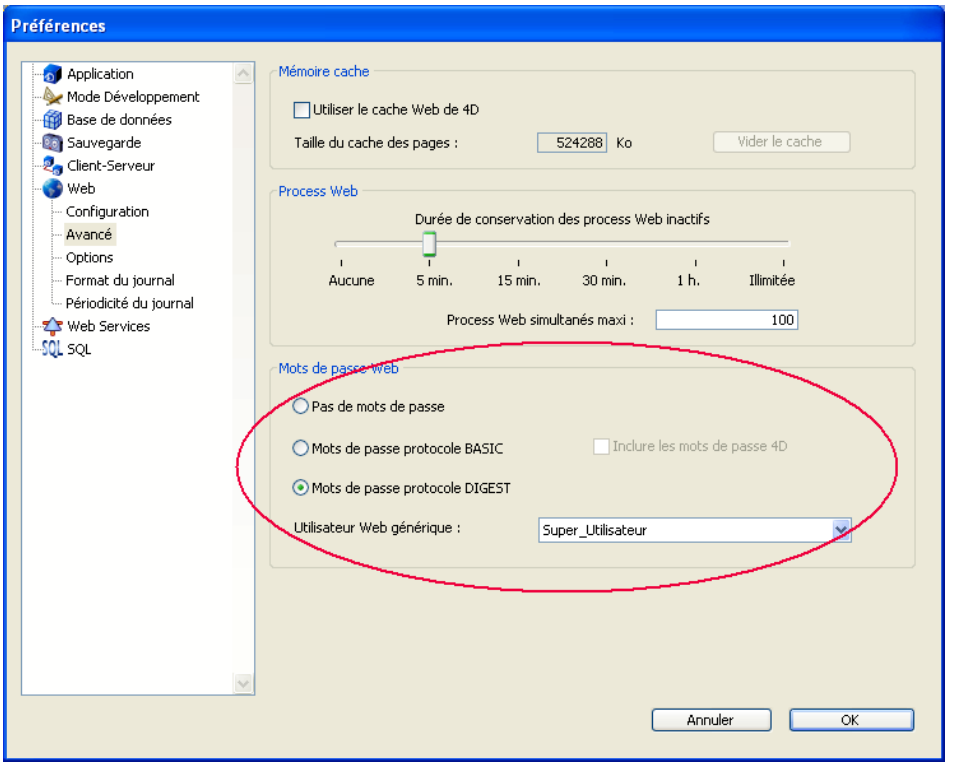
Mode BASIC et Mode DIGEST

Vous pouvez définir, dans les Préférences de la base, le système de contrôle d'accès que vous souhaitez appliquer à votre serveur Web. Deux modes d'authentification sont proposés : le mode BASIC et le mode DIGEST (à compter la v11). Le mode d'authentification concerne la manière dont sont collectées et traitées les informations relatives au nom d'utilisateur et au mot de passe.
• En mode BASIC, le nom et le mot de passe saisis par l'utilisateur sont envoyés en clair dans les requêtes HTTP. Ce principe n'assure pas une sécurité totale au système dans la mesure où ces informations peuvent être interceptées et utilisées par un tiers.
• Le mode DIGEST procure un niveau de sécurité plus élevé car les informations d'authentification sont traitées par un processus unidirectionnel appelé hachage qui rend leur contenu impossible à décrypter.

Côté utilisateur, l'emploi d'un mode d'authentification ou de l'autre est transparente.

Notes :
- Pour des raisons de compatibilité, le mode d'authentification BASIC est utilisé par défaut dans les bases 4D converties en version 11 (si l'option "Utiliser mots de passe" était cochée dans la version précédente). Vous devez activer explicitement le mode Digest.
- L'authentification Digest est une fonction de HTTP1.1 et n'est pas prise en charge par tous les navigateurs. Par exemple, seules les versions 5.0 et suivantes du navigateur Microsoft Internet Explorer acceptent ce mode. Si un navigateur ne prenant pas en charge cette fonctionnalité adresse une demande au serveur Web alors que l'authentification Digest est activée, le serveur rejette la demande et retourne un message d'erreur au navigateur.

Le choix du mode d'authentification s'effectue dans la page **Web/Avancé** de la boîte de dialogue des Préférences :



Dans la zone "Mots de passe", vous disposez de trois options :

• **Pas de mots de passe** : aucune authentification n'est effectuée pour la connexion au serveur Web. Dans ce cas :
- Si la Méthode base Sur authentification Web existe, elle est exécutée et, outre $1 et $2, seules les adresses IP du navigateur et du serveur ($3 et $4) sont renseignées, le nom d'utilisateur et le mot de passe ($5 et $6) sont vides. Vous pouvez dans ce cas filtrer les connexions en fonction de l'adresse IP du navigateur et/ou de l'adresse IP demandée du serveur.
- Si la Méthode base Sur authentification Web n'existe pas, les connexions sont automatiquement acceptées.
• **Mots de passe protocole BASIC** : authentification standard en mode BASIC. Lorsqu'un utilisateur se connecte, une boîte de dialogue lui permettant de saisir son nom et son mot de passe s'affiche sur son navigateur. Ces deux valeurs sont ensuite envoyées en clair à la Méthode base Sur authentification Web avec les autres paramètres de la connexion (adresse et port IP, URL...) pour que vous puissiez les traiter.
Ce mode donne accès à l'option **Inclure les mots de passe 4D**, permettant d'utiliser, au lieu ou en plus de votre propre système, le système 4D de mots de passe de la base (défini dans 4D).

• **Mots de passe protocole DIGEST** : authentification en mode DIGEST. Comme en mode BASIC, l'utilisateur doit saisir son nom et son mot de passe lorsqu'il se connecte. Ces deux valeurs sont alors envoyées cryptées à la Méthode base Sur authentification Web avec les autres paramètres de la connexion. Vous devez authentifier l'utilisateur à l'aide de la commande Valider mot de passe digest Web.

Notes :
- Vous devez redémarrer le serveur Web pour que les modifications effectuées sur ces paramètres soient prises en compte
- Avec le serveur Web de 4D Client, il est à noter que tous les sites publiés par les postes 4D Client partageront la même table d'utilisateurs. En effet, la validation des utilisateurs/mots de passe est effectuée par l'application 4D Server.

Mode BASIC : Combinaison des mots de passe et de la Méthode base Sur authentification Web

Si vous utilisez le mode BASIC, le système de filtrage des connexions au serveur Web de 4D dépend de la combinaison de deux paramètres :
• les options de mots de passe Web dans la boîte de dialogue des Préférences de la base,
• l'existence ou non de la Méthode base Sur authentification Web.

Voici les différentes possibilités de contrôle des connexions :

L'option "Mots de passe protocole BASIC" est cochée et l'option "Inclure les mots de passe de 4D" n'est pas cochée
• Si la Méthode base Sur authentification Web existe, elle est exécutée et tous ses paramètres sont renseignés. Vous pouvez dans ce cas filtrer très précisément les connexions à partir du nom d'utilisateur, du mot de passe et/ou des adresses IP du navigateur et du serveur Web.
• Si la Méthode base Sur authentification Web n'existe pas, la connexion est automatiquement refusée et un message indiquant que la méthode d'authentification n'existe pas est envoyé au navigateur.
Note : Si le nom d'utilisateur envoyé est une chaîne vide et si la Méthode base Sur authentification Web n'existe pas, une boîte de dialogue de demande de mot de passe est envoyée au navigateur.

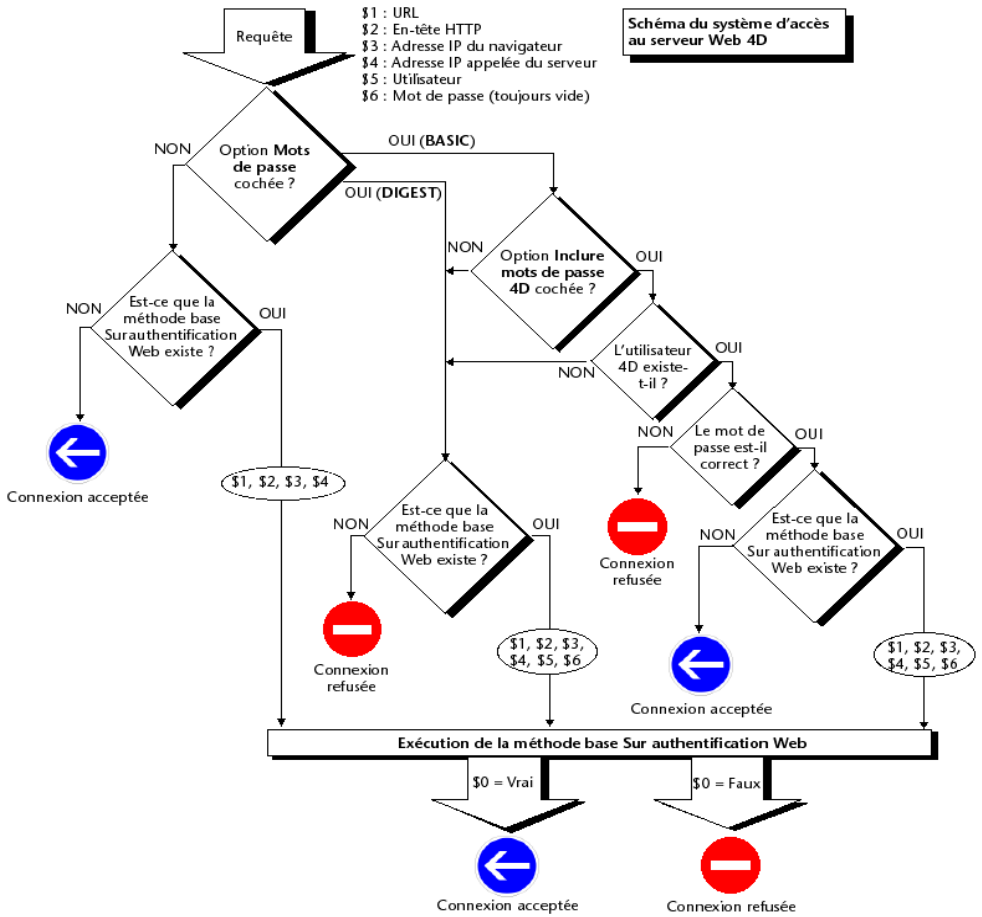
Les options "Mots de passe protocole BASIC" et "Inclure les mots de passe de 4D" sont cochées
• Si le nom d'utilisateur envoyé par le navigateur existe dans la table des utilisateurs 4D et que le mot de passe est valide, la connexion est acceptée (dans ce cas, pour des raisons de sécurité le paramètre $6 n'est alors pas renseigné). Si le mot de passe est invalide, la connexion est refusée.
• Si le nom d'utilisateur envoyé par le navigateur n'existe pas dans 4D, deux cas sont alors possibles :
- si la Méthode base Sur authentification Web existe, les paramètres $1, $2, $3, $4, $5 et $6 sont renseignés. Vous pouvez dans ce cas filtrer très précisément les connexions à partir du nom d'utilisateur, du mot de passe et/ou des adresses IP du navigateur et du serveur Web.
- si la Méthode base Sur authentification Web n'existe pas, la connexion est refusée.

Mode DIGEST

A la différence du mode BASIC, le mode DIGEST n'est pas compatible avec les mots de passe 4D standard : il n'est pas possible d'utiliser les mots de passe 4D comme identifiants Web. L'option "Inclure les mots de passe 4D" est grisée lorsque ce mode est sélectionné. Les identifiants des utilisateurs Web doivent être gérés de façon personnalisée (par exemple via une table).
Lorsque le mode DIGEST est activé, le paramètre $6 (mot de passe) est toujours retourné vide dans la Méthode base Sur authentification Web. En effet dans ce mode, cette information ne transite pas en clair par le réseau. Vous devez impérativement dans ce cas évaluer la demande de connexion à l'aide de la commande Valider mot de passe digest Web.

vous utilisez le mode BASIC, le système de filtrage des connexions au serveur Web de 4D dépend de la

Le fonctionnement du système d'accès au serveur Web 4D est résumé dans le schéma suivant :



A propos des robots (note de sécurité)

Certains robots (moteurs de recherche, spiders) parcourent les serveurs Web et les pages statiques. Si vous souhaitez que les robots ne puissent pas accéder à la totalité de votre site, il est possible de définir des URL qui leur seront interdits.
Pour cela, placez un fichier nommé ROBOTS.TXT à la racine du serveur. Ce fichier doit être structuré de la manière suivante :

User-Agent: <nom>
Disallow: <URL> ou <début d'URL>

Par exemple :

User-Agent: *
Disallow: /4D
Disallow: /%23%23
Disallow: /GIFS/

"User-Agent: *" signifie qu'il s'agit de tous les robots.
"Disallow: /4D" signifie que les robots ne doivent pas accéder aux URL commençant par /4D.
"Disallow: /%23%23" signifie que les robots ne doivent pas accéder aux URL commençant par /%23%23.
"Disallow: /GIFS/' signifie que les robots ne doivent pas accéder au dossier /GIFS/ ni aux sous-dossiers.

Autre exemple :

User-Agent: *
Disallow: /

Dans ce cas, la totalité du site est interdite aux robots.

Utilisateur Web générique

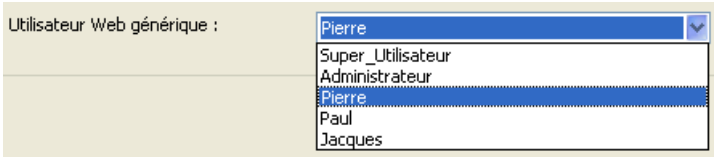
Vous pouvez désigner un utilisateur — préalablement défini dans la table des mots de passe de 4D — comme "Utilisateur Web générique". Dans ce cas, chaque navigateur se connectant à la base bénéficie des autorisations et restrictions d'accès associées à cet utilisateur. Vous pouvez ainsi contrôler simplement l'accès des navigateurs aux différentes parties de la base.

Note : Il ne faut pas confondre cette option, permettant de restreindre les accès des navigateurs aux différentes parties de la base (tables, menus, etc.), avec le système de contrôle des connexions au serveur Web, géré par les mots de passe et la Méthode base Sur authentification Web.

Pour définir un Utilisateur Web générique :

1. En mode Développement, créez au moins un utilisateur dans l'Editeur de mots de passe. Vous pouvez lui associer ou non un mot de passe.
2. Dans les différents éditeurs de 4D, assignez à cet utilisateur les autorisations et restrictions d'accès souhaitées.

3. Dans la boîte de dialogue des Préférences, choisissez le thème **Web**, page **Avancé**.
La zone “Mots de passe Web” contient la liste déroulante **Utilisateur Web générique**. Par défaut, l’utilisateur Web générique est le Super_Utilisateur : les navigateurs disposent donc d’un accès libre à toutes les parties de la base.
4. Choisissez l’utilisateur dans la liste déroulante et validez la boîte de dialogue.



Tous les navigateurs Web autorisés à se connecter à la base bénéficieront des autorisations et restrictions d’accès associées à l’utilisateur Web générique (sauf lorsque le mode BASIC et l’option “Inclure les mots de passe 4D” sont cochés et que l’utilisateur qui se connecte existe dans la table des mots de passe 4D, cf. ci-dessous).

Interaction avec le protocole BASIC

L’option “Mots de passe protocole BASIC” n’influe pas sur le mécanisme de l’utilisateur Web générique : quel que soit l’état de cette option, les privilèges et restrictions d’accès associés à l’“Utilisateur Web générique” seront appliqués à tous les navigateurs Web autorisés à se connecter à la base.

En revanche, lorsque l’option "Inclure les mots de passe 4D" est cochée, deux cas peuvent se produire :
• Le nom et le mot de passe de l’utilisateur n’existent pas dans la table des mots de passe de 4D. Dans ce cas, si la connexion est acceptée par la Méthode base Sur authentification Web, les droits d’accès de l’utilisateur Web générique seront appliqués au navigateur.
• Le nom et le mot de passe de l’utilisateur existent dans la table des mots de passe de 4D. Dans ce cas, le paramètre “Utilisateur Web générique” est ignoré : l’utilisateur se connecte avec ses propres droits d’accès.

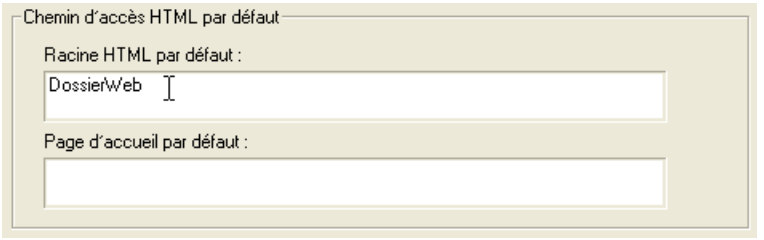
Dossier racine HTML par défaut

Cette option des Préférences de la base vous permet de définir le dossier dans lequel 4D recherchera les pages HTML statiques et semi-dynamiques, les images, etc., à envoyer aux navigateurs.

De plus, le dossier racine HTML définit, sur le disque dur du serveur Web, le niveau hiérarchique au-dessus duquel les fichiers ne seront pas accessibles. Cette restriction d’accès s’applique aux URLs demandés par les navigateurs Web ainsi qu’aux commandes du serveur Web 4D telles que ENVOYER FICHIER HTML. Si un URL demandé par un navigateur ou une commande 4D tente d’accéder à un fichier situé en amont du dossier racine HTML, une erreur est retournée, indiquant que le fichier n’a pas été trouvé.

Par défaut, 4D définit un dossier racine HTML intitulé DossierWeb. S'il n'existe pas déjà, le dossier racine HTML est créé sur le disque au premier lancement du serveur Web.
Si vous conservez l'emplacement par défaut, le dossier racine est créé :
• avec 4D en mode local et 4D Server, au même niveau que le fichier de structure de la base.
• avec 4D en mode distant, dans le dossier local de la base 4D (cf. commande Dossier 4D).

Vous pouvez modifier le nom et l'emplacement du dossier racine HTML par défaut dans la boîte de dialogue des Préférences (thème **Web**, page **Configuration**) :



Dans la zone “Racine HTML par défaut”, saisissez le nouveau chemin d’accès du dossier que vous souhaitez utiliser.
Le chemin d’accès saisi dans cette boîte de dialogue est relatif : il est établi à partir du dossier contenant la structure de la base (4D en mode local ou 4D Server) ou du dossier contenant l'application ou le progiciel 4D (4D en mode distant).
Afin d’assurer la compatibilité multiplate-forme de vos bases, le serveur Web 4D utilise, pour décrire les chemins d’accès, des conventions d’écriture particulières. Les règles de syntaxe sont les suivantes ;
• les dossiers sont séparés par le caractère /
• le chemin ne doit pas se terminer par /
• pour “remonter” d’un niveau dans la hiérarchie des dossiers, saisissez “..” (point point) devant le nom du dossier,
• le chemin ne doit pas commencer par / (sauf si vous souhaitez que le dossier racine HTML soit le dossier de la base ou du 4D distant, cf. ci-dessous).
Par exemple, si vous souhaitez que le dossier racine HTML soit le sous-dossier “Web”, placé dans le dossier “Base4D”, saisissez Bases4D/Web
Si vous souhaitez que le dossier racine HTML soit le dossier de la base ou du 4D distant, mais que l’accès aux dossiers des niveaux supérieurs soit interdit, saisissez / dans la zone.
Pour que l’accès aux volumes soit totalement libre, laissez la zone “Racine HTML par défaut” vide.

ATTENTION : Si vous ne définissez aucun dossier racine HTML par défaut, le dossier contenant le fichier de structure de la base ou l'application 4D est utilisé. **Dans ce cas, il n’y a pas de restrictions d’accès** (tous les volumes sont accessibles).

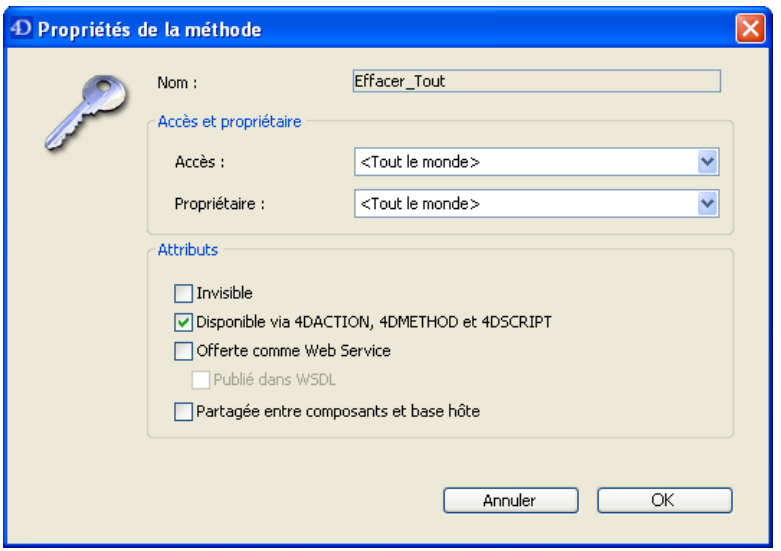
Notes :
• Lorsque le dossier racine HTML est modifié dans les Préférences de la base, le cache est effacé afin de ne pas conserver des fichiers dont l'accès serait devenu restreint.

• Il est possible de définir dynamiquement le dossier racine HTML par défaut à l'aide de la commande FIXER RACINE HTML. Dans ce cas, la modification s'applique à tous les process Web courants pour la session de travail. Le cache des pages HTML est alors effacé.

Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT

Les URLs spéciaux 4DACTION (mode sans contexte) et 4DMETHOD (mode contextuel) ainsi que les balises 4DSCRIPT, 4DVAR et 4DHTMLVAR permettent de déclencher l'exécution de toute méthode projet d'une base 4D publiée sur le Web. Par exemple, la requête http://www.serveur.com/4DACTION/Effacer_Tout provoque l'exécution de la méthode projet *Effacer_Tout*, si elle existe.

Ce mécanisme présente donc un risque pour la sécurité de la base, notamment si un internaute déclenche intentionnellement ou par erreur une méthode non destinée à une exécution via le Web. Vous pouvez prévenir ce risque de trois manières :
• restreindre les accès aux méthodes projet via le système de mots de passe 4D. Inconvénients : ce système nécessite l'utilisation des mots de passe 4D et interdit tout type d'exécution de la méthode (y compris via des balises HTML).
• filtrer les méthodes appelées via des URLs à l'aide de la Méthode base Sur authentification Web. Inconvénients : si la base comporte de nombreuses méthodes, ce système peut être difficile à gérer.
• utiliser l'option **Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT** affichée dans la boîte de dialogue des Propriétés des méthodes projet :



Cette option permet de désigner individuellement chaque méthode projet pouvant être appelée via les URLs spéciaux 4DACTION, 4DMETHOD et les balises 4DSCRIPT, 4DVAR et 4DHTMLVAR. Lorsqu'elle est désélectionnée, la méthode projet concernée ne peut pas être exécutée via une requête HTTP contenant un URL ou une balise spécial(e). En revanche, elle peut être exécutée à l'aide d'autres types d'appels (formules, autres méthodes, etc.).

Cette option est désélectionnée par défaut à la création d'une base. Vous devez désigner expressément les méthodes pouvant être exécutées via les URLs 4DACTION et 4DMETHOD ainsi que les balises 4DSCRIPT, 4DVAR et 4DHTMLVAR.

Dans l'Explorateur, les méthodes projet "disponibles via 4DACTION, 4DMETHOD et 4DSCRIPT" bénéficient d'une icône spécifique :

Référence

Méthode base Sur authentification Web, Méthode base Sur connexion Web, Utiliser le protocole SSL.

La Méthode base Sur authentification Web est chargée de gérer les accès au moteur de serveur Web. Elle est automatiquement appelée par 4D ou 4D Server lorsqu'une requête d'un navigateur Web requiert l'exécution d'une méthode 4D sur le serveur (appel d'une méthode via un URL 4DACTION ou 4DCGI, une balise 4DSCRIPT, etc.).

La méthode base Sur authentification Web reçoit six paramètres de type Texte, passés par 4D ($1, $2, $3, $4, $5 et $6), et retourne un booléen, $0. Voici la description de ces paramètres :

Paramètres	Type	Description
$1	Texte	URL
$2	Texte	En-tête + Corps HTTP (dans la limite des 32 ko)
$3	Texte	Adresse IP du navigateur
$4	Texte	Adresse IP appelée du serveur
$5	Texte	Nom d'utilisateur
$6	Texte	Mot de passe
$0	Booléen	Vrai = requête acceptée, Faux = requête rejetée

Vous devez déclarer ces paramètres de la manière suivante :

```
` Méthode base Sur authentification Web

C_TEXTE($1;$2;$3;$4;$5;$6)
C_BOOLEEN($0)

` Code pour la méthode
```

Note : Tous les paramètres de la Méthode base Sur authentification Web ne sont pas forcément remplis. Les informations reçues par la méthode base dépendent des options que vous avez sélectionnées dans la boîte de dialogue des Préférences de la base. Référez-vous à la section Sécurité des connexions.

- **URL**

Le premier paramètre ($1) est l'URL saisi par l'utilisateur dans la zone 'Adresse' de son navigateur Web, moins l'adresse hôte.
Prenons l'exemple d'une connexion Intranet. Supposons que l'adresse IP de votre machine serveur Web 4D est 123.4.567.89.

Le tableau suivant liste les valeurs de $1 selon l'URL saisi dans le navigateur Web :

URL saisi dans le navigateur Web	Valeur du paramètre $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients/Ajouter	/Clients/Ajouter
123.4.567.89/Faire_ceci/Si_OK/Faire_cela	/Faire_ceci/Si_OK/Faire_cela

• En-tête et corps de la requête HTTP

Le deuxième paramètre ($2) est l'en-tête et le corps de la requête HTTP envoyée par le navigateur Web. Notez que ces informations sont passées telles quelles à la Méthode base Sur authentification Web. Le contenu varie en fonction du type de navigateur Web qui tente de se connecter. Si votre application exploite ces informations, il est de votre ressort d'analyser l'en-tête et le corps.

Note : Pour plus d'informations sur ce paramètre, reportez-vous à la description de la Méthode base Sur connexion Web.

• Adresse IP du navigateur

Le troisième paramètre ($3) reçoit l'adresse IP de la machine du navigateur. Cette information peut vous permettre, en particulier, de distinguer les connexions Intranet des connexions Internet.

• Adresse IP demandée du serveur

Le quatrième paramètre ($4) reçoit l'adresse IP demandée du serveur Web 4D. En effet, 4D autorise le multi-homing, permettant d'exploiter des machines disposant de plusieurs adresses IP. Pour plus d'informations sur ce point, reportez-vous à la section Paramétrages du serveur Web.

• Nom d'utilisateur et Mot de passe

Les paramètres $5 et $6 reçoivent le nom d'utilisateur et le mot de passe saisis par l'utilisateur dans la boîte de dialogue standard d'identification affichée par le navigateur, le cas échéant.
Cette boîte de dialogue apparaît pour chaque connexion dès qu'une option de gestion des mots de passe est cochée dans les Préférences de la base (cf. section Sécurité des connexions).

Note : Si le nom d'utilisateur envoyé par le navigateur existe dans 4D, pour des raisons de confidentialité le paramètre $6 n'est alors pas rempli (il reçoit une chaîne vide).

• Paramètre $0

La Méthode base Sur authentification Web retourne un booléen dans $0 :
• Si $0 est Vrai, la connexion est acceptée.
• Si $0 est Faux, la connexion est refusée.

La Méthode base Sur connexion Web n'est exécutée que si la connexion est acceptée par Sur authentification Web.

ATTENTION : Si aucune valeur n'est passée dans $0, ou si $0 n'est pas définie dans la Méthode base Sur authentification Web, la connexion sera considérée comme **acceptée**, et la Méthode base Sur connexion Web sera exécutée.

Notes

• N'appelez aucun élément d'interface dans la Méthode base Sur authentification Web (ALERTE, DIALOGUE, etc.), sinon son exécution sera interrompue et la connexion refusée. Il en est de même si une erreur se produit durant son traitement.
• Il est possible d'interdire l'exécution par 4DACTION ou 4DMETHOD de chaque méthode projet à l'aide de l'option "Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT" dans la boîte de dialogue des Propriétés des méthodes. Pour plus d'informations sur ce point, reportez-vous à la section Sécurité des connexions.

Appels de la Méthode base Sur authentification Web

La Méthode base Sur authentification Web est automatiquement appelée, quel que soit le mode, lorsqu'une requête ou un traitement nécessite l'exécution d'une méthode 4D. Elle est également appelée lorsque le serveur Web reçoit un URL statique invalide (par exemple, si la page statique demandée n'existe pas).

La Méthode base Sur authentification Web est donc appelée dans les cas suivants :
• lorsque 4D reçoit un URL débutant par 4DACTION/
• lorsque 4D reçoit un URL débutant par 4DMETHOD/
• lorsque 4D reçoit un URL débutant par 4DCGI/
• lorsque 4D reçoit un URL demandant une page statique inexistante
• lorsque 4D traite une balise 4DSCRIPT dans une page semi-dynamique
• lorsque 4D traite une balise 4DLOOP basée sur une méthode dans une page semi-dynamique

A noter que la Méthode base Sur authentification Web n'est PAS appelée lorsque le serveur reçoit un URL demandant une page statique valide.

Exemples

(1) Exemple de Méthode base Sur authentification Web en mode BASIC :

```
    `Méthode base Sur authentification Web
C_TEXTE($5;$6;$3;$4)
C_TEXTE($utilisateur;$motPasse;$IPBrowser;$IPServer)
C_BOOLEEN($utilisateur4D)
TABLEAU TEXTE($utilisateurs;0)
TABLEAU ENTIER LONG($nums;0)
C_ENTIER LONG($upos)
C_BOOLEEN($0)
```

```
$0:=Faux

$utilisateur:=$5
$motPasse:=$6
$IPBrowser:=$3
$IPServer:=$4

    `Pour des raisons de sécurité, refuser les noms qui contiennent @
Si (AvecJoker($utilisateur) | AvecJoker($motPasse))
    $0:=Faux
        `La méthode AvecJoker est décrite ci-dessous
Sinon
        `Vérifier si c'est un utilisateur 4D
    LIRE LISTE UTILISATEURS($utilisateurs;$nums)
    $upos:=Chercher dans tableau($utilisateurs;$utilisateur)
    Si ($upos > 0)
        $utilisateur4D:=Non(Utilisateur supprime($nums{$upos}))
    Sinon
        $utilisateur4D:=Faux
    Fin de si

    Si (Non($utilisateur4D))
            `Ce n'est pas un utilisateur défini dans 4D, chercher dans la table des
                                                                    utilisateurs Web
        CHERCHER([WebUsers];[WebUsers]User=$utilisateur;*)
        CHERCHER([WebUsers]; & [WebUsers]Password=$motPasse)
        $0:=(Enregistrements trouves([WebUsers]) = 1)
    Sinon
        $0:=Vrai
    Fin de si
Fin de si
    `Est-ce une connexion intranet ?
Si (Sous chaine($IPBrowser;1;7) # "192.100.")
    $0:=Faux
Fin de si
```

(2) Exemple de méthode base Sur authentification Web en mode DIGEST :

```
    ` Méthode base Sur authentification Web
C_TEXTE($1;$2;$5;$6;$3;$4)
C_TEXTE($utilisateur)
C_BOOLEEN($0)
$0:=Faux
$utilisateur:=$5
```

```
    `Pour des raisons de sécurité, refuser les noms qui contiennent @
Si (AvecJoker($utilisateur))
$0:=Faux
    `La méthode AvecJoker est décrite ci-dessous
Sinon
    CHERCHER([WebUsers];[WebUsers]User=$utilisateur)
    Si (OK=1)
        $0:=Valider mot de passe digest Web($utilisateur;[WebUsers]Mdp)
    Sinon
        $0:=Faux `Utilisateur inexistant
    Fin de si
Fin de si
```

La Méthode projet *AvecJoker* est décrite ci-dessous :

```
    `Méthode projet AvecJoker
    `AvecJoker ( Chaine ) -> Booléen
    `AvecJoker ( Nom ) -> Contient un joker

C_ENTIER($i)
C_BOOLEEN($0)
C_TEXTE($1)
$0:=Faux
Boucle($i;1;Longueur($1))
    Si (Code de caractere(Sous chaine($1;$i;1)) = Code de caractere("@"))
        $0:=Vrai
    Fin de si
Fin de boucle
```

Référence

Méthode base Sur connexion Web, Présentation des méthodes base, Sécurité des connexions, URLs et actions de formulaires.

La Méthode base Sur connexion Web peut être appelée dans trois cas distincts :
• le serveur Web reçoit une requête débutant par l'URL 4DCGI.
• le serveur Web reçoit une requête invalide.
• elle est également appelée par 4D (mode local) et 4D Server chaque fois qu'un navigateur Web se connecte à la base en mode contextuel ou qu'une requête nécessite la création d'un contexte (ce cas n'est pas géré par 4D en mode distant qui ne prend pas en charge le mode contextuel).

Pour plus d'informations, reportez-vous ci-dessous au paragraphe “Appels de la Méthode base Sur connexion Web”.

La requête doit auparavant avoir été "acceptée" par la Méthode base Sur authentification Web (si elle existe), et la base doit être publiée en tant que serveur Web.

La Méthode base Sur connexion Web reçoit six paramètres de type Texte, passés par 4D ($1, $2, $3, $4, $5 et $6). Voici leur description :

Paramètres	Type	Description
$1	Texte	URL
$2	Texte	En-tête + corps HTTP
$3	Texte	Adresse IP du navigateur
$4	Texte	Adresse IP appelée du serveur
$5	Texte	Nom d’utilisateur
$6	Texte	Mot de passe

Vous devez déclarer ces six paramètres de la manière suivante :

```
` Méthode base Sur connexion Web

C_TEXTE($1;$2;$3;$4;$5;$6)

` Code pour la méthode
```

• Données supplémentaires de l’URL

Le premier paramètre ($1) est l’URL saisi par l'utilisateur dans la zone 'Adresse' de son navigateur Web, moins l’adresse hôte.

Prenons l'exemple d'une connexion Intranet. Supposons que l'adresse IP de votre machine serveur Web 4D est 123.4.567.89. Le tableau suivant liste les valeurs de $1 selon l'URL saisi dans le navigateur Web :

URL saisi dans le navigateur	Valeur du paramètre $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients	/Clients
http://123.4.567.89/Clients/Ajouter	/Clients/Ajouter
123.4.567.89/Faire_ceci/Si_OK/Faire_cela	/Faire_ceci/Si_OK/Faire_cela

Notez que vous êtes libre d'utiliser ce paramètre à votre convenance. 4D ignore simplement les valeurs passées au-delà de la partie hôte de l'URL. Par exemple, vous pouvez établir une convention dans laquelle la valeur "/Clients/Ajouter" signifie "aller directement à l'ajout d'un nouvel enregistrement dans la table [Clients]." En fournissant aux utilisateurs Web de votre base une liste des valeurs possibles et/ou des marqueurs par défaut, vous pouvez proposer des raccourcis vers différentes parties de votre application. Ainsi, les utilisateurs Web peuvent accéder rapidement aux ressources de votre site Web sans devoir naviguer dans toute la base à chaque connexion.

ATTENTION : Pour empêcher un utilisateur d'accéder directement à une base à l'aide d'un marqueur créé pendant une session précédente, 4D intercepte tout URL correspondant à un des URLs standard de 4D.

• En-tête et corps de la requête HTTP

Le deuxième paramètre ($2) est l'en-tête suivi du corps de la requête HTTP envoyée par le navigateur Web. Notez que ces informations sont passées telles quelles à votre Méthode base Sur connexion Web. Le contenu varie en fonction du type de navigateur Web qui tente de se connecter.

Avec Netscape 4.5 sous Mac OS, vous recevrez un en-tête semblable à celui-ci :

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.5 (Macintosh; I; PPC)
Host: 123.45.67.89
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: fr
Accept-Charset: iso-8
```

Avec Microsoft Internet Explorer 6 sous Windows, vous recevrez un en-tête semblable à celui-ci :

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 123.45.67.89
Accept: image/gif, image/x-xbitmap, image/pjpeg, */*
Accept-Language: fr
```

Si votre application exploite ces informations, il est de votre ressort d'analyser l'en-tête et le corps.

• Adresse IP du navigateur

Le troisième paramètre ($3) reçoit l'adresse IP de la machine du navigateur. Cette information peut vous permettre, en particulier, de distinguer les connexions Intranet des connexions Internet.

• Adresse IP demandée du serveur

Le quatrième paramètre ($4) reçoit l'adresse IP demandée du serveur Web 4D. En effet, 4D autorise le multi-homing, permettant d'exploiter des machines disposant de plusieurs adresses IP. Pour plus d'informations sur ce point, reportez-vous à la section Paramétrages du serveur Web.

• Nom d'utilisateur et Mot de passe

Les paramètres $5 et $6 reçoivent le nom d'utilisateur et le mot de passe saisis par l'utilisateur dans la boîte de dialogue standard d'identification affichée par le navigateur, le cas échéant. Cette boîte de dialogue apparaît pour chaque connexion dès que l'option **Utiliser les mots de passe** est cochée dans les Préférences (cf. section Sécurité des connexions).

Note : Si le nom d'utilisateur envoyé par le navigateur existe dans 4D, pour des raisons de confidentialité le paramètre $6 n'est alors pas rempli (il reçoit une chaîne vide).

Appels de la Méthode base Sur connexion Web

La Méthode base Sur connexion Web peut être utilisée comme point d'entrée dans le serveur Web 4D, soit à l'aide de l'URL spécial 4DCGI, soit à l'aide d'URLs de commande personnalisés. Elle joue aussi le rôle de point d'entrée en mode contextuel (avec 4D en mode local et 4D Server).

Attention : L'appel d'une commande 4D affichant un élément d'interface (ALERTE, DIALOGUE...) entraîne l'arrêt du traitement de la méthode.

La Méthode base Sur connexion Web est donc appelée dans les cas suivants :
• lors de la connexion d'un navigateur à un serveur Web 4D fonctionnant en mode contextuel. La méthode base est appelée avec l'URL /<action>...
• lorsque 4D reçoit l'URL /4DMETHOD. Le serveur Web passe en mode contextuel et la méthode base est appelée avec l'URL /4DMETHOD/NomMéthode dans $1.
• lorsque 4D reçoit l'URL /4DCGI. La méthode base est appelée avec l'URL /4DCGI/<action> dans $1.
• lorsqu'une page Web appelée avec un URL du type <chemin>/<fichier> n'est pas trouvée. La méthode base est appelée avec l'URL (*).
• lorsqu'une page Web est appelée avec un URL du type <chemin>/ et qu'aucune page d'accueil par défaut n'est définie. La méthode base est appelée avec l'URL (*).

(*) Dans ces cas particuliers, l'URL reçu dans $1 ne débute pas par le caractère "/".

Pour connaître l'origine de l'appel de la Méthode base Sur connexion Web et adapter le traitement à effectuer, vous devez utiliser la fonction Contexte Web, qui retourne Vrai si elle est appelée depuis le mode contextuel, et Faux sinon.
Par conséquent, lorsque votre serveur Web manipule le mode contextuel et le mode sans contexte, nous vous conseillons de structurer la Méthode base Sur connexion Web de la manière suivante :

```
    `Méthode base Sur connexion Web
C_TEXTE($1;$2;$3;$4;$5;$6)
Si (Contexte Web)  `Si l'on est en mode contextuel
    AvecContexte ($1;$2;$3;$4;$5;$6)
        `La méthode AvecContexte contient tout ce qu'il y avait dans la
        `méthode base Sur connexion Web en 4D 6.0.x
Sinon
    SansContexte ($1;$2;$3;$4;$5;$6)
        `La méthode SansContexte effectue les traitements des requêtes
        `non contextuelles (généralement courts)
Fin de si
```

Exemple : implémentation des pages Home locales des clients en mode contextuel

Dans l'exemple suivant, le paramètre $1, envoyé à la méthode base Sur connexion Web, est utilisé pour implémenter les pages Home des clients dans une entreprise. Le serveur Intranet fonctionne en mode contextuel.

La base contient deux tables : [Clients] et [Tables]. La Méthode base Sur ouverture ci-dessous initialise les tableaux interprocess utilisés ultérieurement par la Méthode base Sur connexion Web.

```
    ` Méthode base Sur ouverture

    ` Liste des tables
TABLEAU ALPHA(31;<>taTables;Lire numero derniere table)
Boucle ($vlTable;1;Taille tableau(<>taTables);1;-1)
    Si(Est un numero de table valide($vlTable))
        <>taTables{$vlTable}:=Nom de la table($vlTable)
    Sinon
        SUPPRIMER DANS TABLEAU(<>taTables;$vlTable)
    Fin de si
Fin de boucle

    ` Actions Web standard à la connexion
TABLEAU ALPHA(31;<>taActions;2)
<>taActions{1}:="Ajouter"
<>taActions{2}:="Visualiser"
```

Le principal objectif de la méthode base Sur connexion Web est de déchiffrer les données de l'URL situées après l'adresse de l'hôte et d'agir en conséquence. La méthode est la suivante :

```
    ` Méthode base Sur connexion Web

C_TEXTE($1;$2;$3;$4;$5;$6)
C_TEXTE($vtURL)

Si (Contexte Web)   ` Si nous sommes bien en mode contextuel
        ` Par précaution, vérifiez que $1 est égal à "/" ou "/..."
    Si ($1="/@")
            ` Copier l'URL dans une variable locale moins le premier "/"
        $vtURL:=Sous chaine($1;2)
            ` Analyser l'URL et remplir un tableau local avec les informations de l'URL
            ` Par exemple, si les données supplémentaires de l'URL sont "aaa/bbb/ccc", le
            ` tableau contiendra les trois éléments "aaa", "bbb" et "ccc", dans cet ordre
        $vlElem:=0
        TABLEAU TEXTE($atTokens;$vlElem)
```

```
Tant que ($vtURL # "")
    $vlElem:=$vlElem+1
    INSERER DANS TABLEAU($atTokens;$vlElem)
    $vlPos:=Position("/";$vtURL)
    Si ($vlPos>0)
        $atTokens{$vlElem}:=Sous chaine($vtURL;1;$vlPos-1)
        $vtURL:=Sous chaine($vtURL;$vlPos+1)
    Sinon
        $atTokens{$vlElem}:=$vtURL
        $vtURL:=""
    Fin de si
Fin tant que
    ` Si des données supplémentaires sont passées au-delà de la partie hôte de l'URL
Si ($vlElem>0)
        ` Utiliser le tableau interprocess initialisé dans la méthode base Sur
        ` ouverture. Vérifier si la première information est un nom de table
    $vlTableNumber:=Chercher dans tableau(<>taTables;$atTokens{1})
    Si ($vlTableNumber>0)
            ` Si oui, obtenir un pointeur vers cette table
        $vpTable:=Table($vlTableNumber)
            ` Définir les formulaires entrée et sortie
        FORMULAIRE ENTREE($vpTable->;"Input Web")
        FORMULAIRE SORTIE($vpTable->;"Output Web")
            ` Utiliser le tableau interprocess initialisé dans la méthode base Sur
                                                                    ouverture
            ` Vérifier si la deuxième information est une action standard connue
        $vlAction:=Chercher dans tableau(<>taActions;$atTokens{2})
        Au cas ou
                ` Ajouter des enregistrements
            : ($vlAction=1)
                Repeter
                    AJOUTER ENREGISTREMENT($vpTable->;*)
                Jusque (OK=0)
                ` Visualiser les enregistrements
            : ($vlAction=2)
                LECTURE SEULEMENT($vpTable->)
                TOUT SELECTIONNER($vpTable->)
                VISUALISER SELECTION($vpTable->;*)
                LECTURE ECRITURE($vpTable->)
```

```
                    Sinon
                        ` Ajouter éventuellement ici des actions standard supplémentaires à
                                                                    implémenter
                    Fin de cas
                Sinon
                    ` Ajouter éventuellement ici d'autres actions standard à implémenter
                Fin de si
            Fin de si
        Fin de si
            ` Dans tous les cas, poursuivre le processus standard de connexion
        WWW CONNEXION STANDARD
    Sinon
        ... ` Placer éventuellement ici le code gérant le mode sans contexte
    Fin de si
```

A ce stade, les membres de l'entreprise peuvent se connecter à la base et saisissent un URL en fonction des conventions définies. Les utilisateurs peuvent également créer des marqueurs “Favoris” s'ils ne veulent pas saisir l'URL à chaque connexion. En fait, la meilleure solution consiste à fournir à chaque membre de l'entreprise une page HTML qu'il peut utiliser en local pour accéder à la base. Voici la page HTML :

Autrement dit, la page HTML Dupont_IS.HTM est la page Home du client local pour le système d'informations de l'entreprise basé sur 4D. Si l'utilisateur clique sur le lien Créer produits, le navigateur Web se connectera à l'hôte dont l'URL est http://123.4.567.89/Produits/Ajout. Si l'adresse IP de l'ordinateur où se trouve la base est 123.4.567.89, la méthode base Sur connexion

Web reçoit les données d'URL supplémentaires "/Produits/Ajout" dans $1 puis lance le processus d'ajout d'enregistrements à la table [Produits].

Enfin, les utilisateurs peuvent glisser-déposer les liens de cette page vers leur bureau pour créer une icône de raccourci Internet, telle que l'icône ci-dessous. Double-cliquer sur une de ces icônes envoie directement l'utilisateur dans la zone spécifiée de votre base 4D Web.

Le code source de cette page HTML est le suivant :

```
<HTML>
<HEAD>
    <TITLE>Système d'information Intranet Dupont SARL</TITLE>
</HEAD>
<BODY>
<H1><B>Système d'information Intranet Dupont SARL</B></H1>
<P ALIGN=CENTER><TABLE BORDER=1 CELLPADDING=1 WIDTH="100%">
    <TR>
    <TD>
        <P ALIGN=CENTER><A HREF="http://123.4.567.89/Clients/Ajout">Créer clients</A>
    </TD>
    <TD>
        <P ALIGN=CENTER><A HREF="http://123.4.567.89/Produits/Ajout">Créer produits</A>
    </TD>
    </TR>
    <TR>
    <TD>
        <P ALIGN=CENTER><A HREF="http://123.4.567.89/Clients/Liste">Visualiser clients</A>
    </TD>
    <TD>
        <P ALIGN=CENTER><A HREF="http://123.4.567.89/Produits/Liste">Visualiser produits</A>
    </TD>
    </TR>
</TABLE></P>
<P><B><A HREF="Help.HTM">Cliquez ici pour obtenir de l'aide</A></B></P>
</BODY>
</HTML>
```

Référence

Méthode base Sur authentification Web, Présentation des méthodes base, URLs et actions de formulaires, Utiliser le mode contextuel.

Cette section décrit les moyens mis à votre disposition par le serveur Web de 4D pour échanger de l'information via le Web, c'est-à-dire pour envoyer et recevoir dynamiquement des valeurs. Les points suivants sont abordés :
• Envoyer des valeurs dynamiques stockées dans des variables 4D
• Recevoir des valeurs dynamiques via des formulaires Web
• Utiliser la méthode projet COMPILER_WEB
• Gérer des images interactives (server-side image mapping)
• Encapsuler du JavaScript

Note : l'envoi et la réception de valeurs dynamiques peuvent être effectués automatiquement en mode contextuel via les formulaires 4D convertis. Pour plus d'informations sur ce point, reportez-vous à la section Utiliser le mode contextuel.

Envoyer des valeurs dynamiques

Vous pouvez insérer dans vos pages HTML des références à des variables 4D. Ces références peuvent être associées à tout type d'objet HTML. Au moment de l'envoi des pages Web aux navigateurs, 4D remplacera ces références par la valeur courante de la variable. Les pages reçues sont alors la combinaison d'éléments statiques et de valeurs issues de 4D. Ce type de page est appelé page semi-dynamique.

Notes :
• Vous devez utiliser des variables process.
• Comme le HTML est orienté Texte, vous travaillerez généralement avec des variables Texte. Toutefois, vous pouvez également utiliser des variables BLOB (ce qui vous permet de vous affranchir de la limite des 32 000 caractères inhérente aux variables de type texte). Il vous suffit pour cela de générer des BLOBs en mode Texte sans longueur.

Tout d'abord, un objet HTML peut voir sa valeur initialisée par la valeur d'une variable 4D. Ensuite, lorsqu'un formulaire Web est envoyé en retour, la valeur d'un objet HTML peut être retournée dans une variable 4D. Pour cela, dans le code du formulaire HTML, vous créez un objet HTML **dont le nom est le même que celui de la variable process 4D** que vous voulez associer. Ce point est étudié plus loin dans le paragraphe “Recevoir des valeurs dynamiques ”.

Note : En mode sans contexte, il n’est pas possible de faire référence à des variables images 4D.

Revenons au premier point. Vous pouvez donner par programmation des valeurs par défaut aux objets HTML en incluant <!--#4DVAR NomVar--> dans, par exemple, le champ **valeur** de l'objet HTML ; NomVar est le nom de la variable process 4D telle qu'elle est définie dans le process Web courant — nom que vous mettez entre <!—# —>, c'est-à-dire la notation standard pour les commentaires HTML.

Note : Certains éditeurs HTML n'acceptent pas la saisie de la valeur <!—#4DVAR NomVar—> dans le champ **valeur** des objets HTML. Dans ce cas, vous devrez la placer directement dans le code HTML.

La balise <!--#4DVAR --> permet également d'insérer des **expressions 4D** dans les pages envoyées (champs, éléments de tableaux, etc.). Le principe de fonctionnement de la balise avec ce type de donnée est identiques à celui des variables. Pour plus d'informations, reportez-vous à la section Balises HTML 4D.

En fait, la syntaxe <!--#4DVAR NomVar--> permet d'insérer des données 4D partout dans la page HTML. Si, par exemple, vous écrivez :

<P>Bienvenue dans <!--#4DVAR vtSiteName-->!</P>

La valeur de la variable 4D vtSiteName sera insérée dans la page HTML.

Examinons un exemple :

```
    ` Voici le code 4D assignant "4D4D" à la variable process vs4D
vs4D:="4D4D"
    ` Puis envoyer la page HTML "AnyPage.HTM"
ENVOYER FICHIER HTML("AnyPage.HTM")
```

Le source de la page HTML AnyPage.HTM est le suivant :

```
<HTML>
<HEAD>
   <TITLE>AnyPage</TITLE>
   <SCRIPT LANGUAGE="JavaScript"><!--

       function Is4DWebServer(){
          return (document.frm.vs4D.value=="4D4D")
       }

       function HandleButton(){
          if (Is4DWebServer()){
             alert("You are connected to 4D Web Server!")
          } else {
             alert("You are NOT connected to 4D Web Server!")
          }
       }


   //--></SCRIPT>
</HEAD>
<BODY>
<FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frm">

<P><INPUT TYPE="hidden" NAME="vs4D" VALUE="<!--4DVAR vs4D-->"</P>

<P><A HREF="JavaScript:HandleButton()"><IMG SRC="AnyGIF.GIF" BORDER=0
ALIGN=bottom></A></P>

<P><INPUT TYPE="submit" NAME="bOK" VALUE="OK"></P>

</FORM>
</BODY>
</HTML>
```

Dans le code HTML ci-dessus, vous remarquez le type de saisie **hidden** dont le nom est vs4D. La valeur de cet objet est mise à la valeur texte "<!--#4DVAR vs4D-->". Comme la méthode projet qui envoie le fichier HTML a préalablement défini la variable process vs4D, 4D remplace la valeur de l'objet HTML et la met à "4D4D", la valeur de la variable 4D.

Vous notez aussi la fonction encapsulée JavaScript Is4DWebServer qui teste la valeur de l'objet HTML vs4D. Voici l'astuce : si la page HTML est envoyée par 4D, la valeur de l'objet est changée en "4D4D" ; en revanche, si la page HTML est envoyée par une autre application (par exemple 4D WebSTAR sur Macintosh), l'objet conserve sa valeur telle qu'elle est définie dans la page, c'est-à-dire "[vs4D]". En testant la valeur de cet objet, par JavaScript, vous pouvez détecter à l'intérieur de la page, côté navigateur Web, si cette page est ou non envoyée par 4D.

Ce premier exemple montre comment vous pouvez construire des pages HTML “intelligentes”, qui comportent des caractéristiques supplémentaires lorsqu'elles sont envoyées par 4D tout en restant compatibles avec d'autres serveurs de Web.

Important : Rappelons que vous ne devez associer que des variables process. De plus, vous ne pouvez pas, dans la version actuelle du programme, associer de tableau 4D à un objet HTML SELECT. En revanche, chaque élément d'un objet SELECT peut se référer à des variables 4D séparées (par exemple, le premier élément à V1, le second à V2, et ainsi de suite).

L'association dans le sens 4D vers navigateur Web fonctionne avec n'importe quel procédé d'encapsulation (ENVOYER FICHIER HTML, ENVOYER BLOB HTML ainsi que, en mode contextuel, texte statique ou variable texte ou BLOB dans un formulaire 4D).

Analyse des pages envoyées par le serveur

• En mode contextuel, avant l'envoi d'une page HTML (document HTML ou formulaire 4D traduit), 4D scrute toujours le source HTML pour rechercher les objets renvoyant à des variables 4D.
• En mode sans contexte, à des fins d'optimisation l'analyse du source HTML n'est pas effectuée par le serveur Web de 4D lorsque les pages HTML sont appelées via des URLs simples et qu'elles sont suffixées ".HTML" ou ".HTM". 4D propose également des mécanismes permettant de "forcer" l'analyse des pages si nécessaire (reportez-vous à la section Balises HTML 4D).

Insérer du code HTML dans des variables 4D

Vous pouvez insérer du code HTML dans des variables 4D incluses dans des pages statiques. Lorsque la page est affichée sur le navigateur, la valeur de la variable est remplacée par le code HTML, qui est alors interprété par le navigateur.

Pour insérer du code HTML dans des variables 4D, vous disposez de deux possibilités :
• Faire débuter la variable 4D par le code 1 (par exemple vtHTML:=Caractere(1)+"...Code HTML...") et l'insérer dans la page HTML via la balise <!--#4DVAR vtHTML-->.
• Insérer une variable 4D standard (par exemple vtHTML:="...Code HTML...") dans la page HTML via la balise <!--#4DHTMLVAR vtHTML-->.
Vous pouvez utiliser une variable de type Texte ou BLOB (généré en mode Texte sans longueur).

Pour plus d'informations, reportez-vous à la section Balises HTML 4D.

Recevoir des valeurs dynamiques

Quand vous envoyez une page HTML avec ENVOYER FICHIER HTML ou ENVOYER BLOB HTML, vous pouvez aussi associer des variables 4D avec des objets HTML dans le sens "navigateur Web vers 4D". L'association fonctionne dans les deux sens : lorsqu'un formulaire HTML est retourné, 4D peut recopier les valeurs des objets HTML dans des variables process 4D. En vue de la compilation de la base, ces variables doivent être déclarées dans la méthode COMPILER_WEB (cf. paragraphe ci-dessous).

Il est également posssible de récupérer les valeurs des formulaires Web envoyés à 4D sans connaître au préalable les champs qu'ils contiennent à l'aide de la commande LIRE VARIABLES FORMULAIRE WEB. Pour plus d'informations, reportez-vous à la description de cette commande.

Attention : La récupération des valeurs dans les variables process 4D n'est possible qu'avec des pages HTML envoyées avec ENVOYER FICHIER HTML ou ENVOYER BLOB HTML. Dans le cas de HTML encapsulé dans un formulaire 4D en mode contextuel, la récupération des valeurs est limitée aux objets 4D effectivement placés dans ce formulaire.

Considérons cette page de code HTML :

```
<HTML>
<HEAD>
   <TITLE>Welcome</TITLE>
   <SCRIPT LANGUAGE="JavaScript"><!--

        function GetBrowserInformation(formObj){
           formObj.vtNav_appName.value = navigator.appName
           formObj.vtNav_appVersion.value = navigator.appVersion
           formObj.vtNav_appCodeName.value = navigator.appCodeName
           formObj.vtNav_userAgent.value = navigator.userAgent
           return true
        }

        function LogOn(formObj){
           if(formObj.vtUserName.value!=""){
              return true
           } else {
              alert("Enter your name, then try again.")
              return false
           }
        }
    //--></SCRIPT>
</HEAD>
<BODY>
<FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frmWelcome"
                        onsubmit="return GetBrowserInformation(frmWelcome)">

<H1>Welcome to Spiders United</H1>

<P><B>Please Enter your Name:</B>
   <INPUT TYPE="text" NAME="vtUserName" VALUE="<!--4DVAR vtUserName-->" SIZE=30></P>
<P>
   <INPUT TYPE="submit" NAME="vsbLogOn" VALUE="Log On" onclick="return LogOn(frmWelcome)">
   <INPUT TYPE="submit" NAME="vsbRegister" VALUE="Register">
   <INPUT TYPE="submit" NAME="vsbInformation" VALUE="Information">
</P>

<P>
   <INPUT TYPE="hidden" NAME="vtNav_appName" VALUE="">
   <INPUT TYPE="hidden" NAME="vtNav_appVersion" VALUE="">
   <INPUT TYPE="hidden" NAME="vtNav_appCodeName" VALUE="">
   <INPUT TYPE="hidden" NAME="vtNav_userAgent" VALUE="">
</P>
</FORM>
</BODY>
</HTML>
```

Lorsque la page est envoyée au navigateur Web par 4D, elle se présente ainsi :

Voici les caractéristiques de cette page :

• Elle comporte trois boutons Submit : vsbLogOn, vsbRegister et vsbInformation.
• Le Submit du formulaire quand vous cliquez sur LogOn est d'abord traité par la fonction JavaScript LogOn. Si aucun nom n'a été saisi, le formulaire n'est pas envoyé à 4D et une alerte JavaScript est affichée.
• Le formulaire comporte une méthode 4D POST et un script Submit (GetBrowserInformation) qui copie les propriétés du Navigator dans quatre objets cachés dont les noms commencent par vtNav_App.
• Vous remarquez aussi l'objet vtUserName dont la valeur initiale est <!--#4DVAR vtUserName-->.

Examinons maintenant la méthode 4D (nommé ici WWW Welcome) qui envoie cette page HTML à l'aide de la commande ENVOYER FICHIER HTML. Cette méthode est appelée par la Méthode base Sur connexion Web.

```
      ` Méthode projet WWW Welcome
      ` WWW Welcome -> Booleén
      ` WWW Welcome -> Oui = La session peut commencer

C_BOOLEEN($0)
$0:=Faux

      ` Objets de saisie HTML cachés retournant l'information sur le Navigateur
C_TEXTE(vtNav_appName;vtNav_appVersion;vtNav_appCodeName;vtNav_userAgent)
vtNav_appName:=""
vtNav_appVersion:=""
vtNav_appCodeName:=""
vtNav_userAgent:=""
```

```
    ` Objet texte saisie HTML où le nom de l'utilisateur est saisi
C_TEXTE(vtUserName)
vtUserName:=""

    ` Valeurs du HTML bouton Submit
C_ALPHA(31;vsbLogOn;vsbRegister;vsbInformation)

Repeter
        ` Ne pas oublier d'initialiser les valeurs des boutons Submit!
    vsbLogOn:=""
    vsbRegister:=""
    vsbInformation:=""
        ` Envoyer la page Web
    ENVOYER FICHIER HTML("Welcome.HTM")
        ` Tester les valeurs des boutons Submit pour détecter si l'un d'entre eux
        ` a reçu un clic souris
    Au cas ou

            ` Le bouton Log On bouton a été cliqué
        : (vsbLogOn # "")
            CHERCHER([WWW Users];[WWW Users]User Name=vtUserName)
            $0:=(Enregistrements trouves([WWW Users])>0)
            Si ($0)
                WWW POST EVENT ("Log On";WWW Log information )
                    ` La méthode WWW POST EVENT sauvegarde l'information
                    ` dans une table de la base
            Sinon
                CONFIRMER("Ce nom d'utilisateur est inconnu. Voulez-vous
                                                                    l'enregistrer?")
                $0:=(OK=1)
                Si ($0)
                    $0:=WWW Register
                        ` La méthode WWW Register permet à un nouvel utilisateur
                        ` Web de s'enregistrer
                Fin de si
            Fin de si

            ` Le bouton Register a reçu un clic souris
        : (vsbRegister # "")
            $0:=WWW Register
```

```
                ` Le bouton Information a reçu un clic souris
            : (vsbInformation # "")
                DIALOGUE([User Interface];"WWW Information")
        Fin de cas

    Jusque (Non(<>vbWebServicesOn) | $0)
```

Voici les caractéristiques de cette méthode :

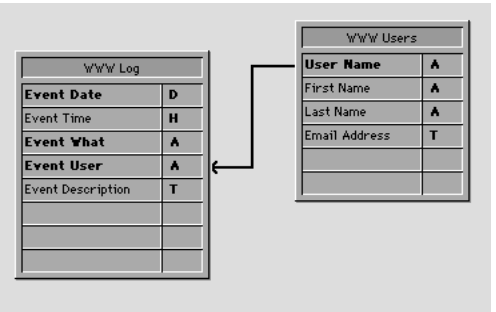
• Les variables 4D vtNav_appName, vtNav_appVersion, vtNav_appCodeName et vtNav_userAgent (liées aux objets HTML de même nom) reçoivent les valeurs assignées aux objets HTML par le script JavaScript GetBrowserInformation. C'est simple : initialisez les variables en chaînes de caractères, et ensuite récupérez les valeurs après que la page ait été soumise.

• Les variables 4D vsbLogOn, vsbRegister et vsbInformation sont liées aux trois boutons Submit. Notez que ces variables sont remises à leur valeur vide chaque fois que la page est envoyée au navigateur. Quand le submit est effectué par l'un de ces trois boutons, le navigateur retourne à 4D la valeur du bouton sur lequel on a cliqué. Comme les variables 4D sont initialisées à chaque fois, la variable qui n'est plus égale à chaîne vide vous indique sur quel bouton on a cliqué. Les deux autres variables sont des chaînes vides, non pas parce que le navigateur a retourné des chaînes vides, mais parce qu'il “n'a rien dit” à leur sujet et, en conséquence, 4D les a laissées inchangées. C'est la raison pour laquelle il est nécessaire de réinitialiser ces variables toutes les fois que la page est envoyée au navigateur.
C'est **de cette manière que vous pouvez savoir** quel bouton Submit a été activé quand plusieurs boutons Submit se trouvent sur la page Web. Notez que les boutons 4D (dans un formulaire 4D) sont des variables numériques. En revanche, en HTML, tous les objets sont des objets texte.

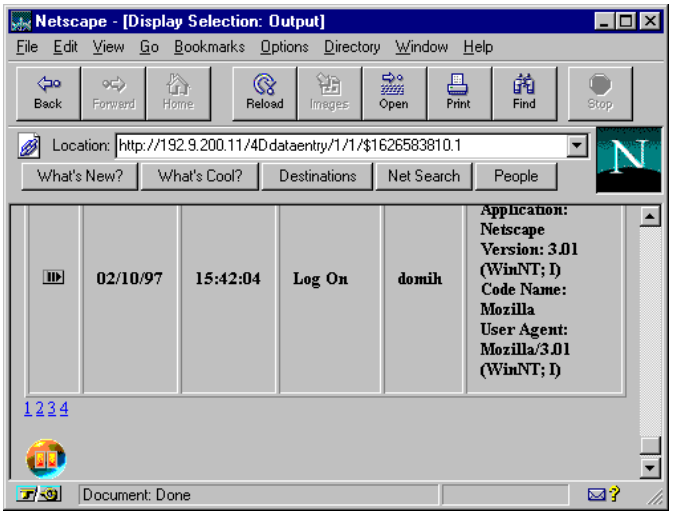
Si vous liez une variable 4D avec un objet SELECT, vous liez aussi une variable texte. Dans 4D, pour tester quel élément d'une liste déroulante a été sélectionné, vous testez la valeur numérique du tableau 4D. En HTML, c'est la valeur de l'élément sélectionné qui est retournée dans la variable 4D liée à l'objet HTML.

Quel que soit l'objet que vous associez à une variable 4D, la valeur retournée est de type Texte. Donc, vous devez associer des variables 4D process de type Alpha ou Texte.

Un point intéressant de cet exemple (partiel) est qu'une fois que vous avez obtenu l'information sur le navigateur, vous pouvez stocker ces valeurs dans une table 4D, et là encore vous associez le Web et les capacités de base de données de 4D. C'est ce que réalise la méthode projet WWW POST EVENT. Elle ne “poste” pas un événement, elle sauvegarde les informations de la session Web dans des tables du type de celles-ci :



Une fois que vous avez sauvegardé ces informations dans une table, vous pouvez, avec d'autres méthodes projet, renvoyer l'information à l'utilisateur Web. Pour cela, vous utilisez simplement un CHERCHER pour trouver la bonne information et un VISUALISER SELECTION pour montrer les enregistrements [WWW Log]. L'image ci-dessous présente les informations de connexion que l'utilisateur Web peut obtenir s'il est un utilisateur enregistré du site Web :



Compte tenu des possibilités des liens 4D/HTML dans cet exemple et des informations que vous pouvez demander aux utilisateurs par des dialogues HTML ou des formulaires 4D, vous pouvez concevoir des capacités d'administration très intéressantes pour votre site Web.

Méthode projet COMPILER_WEB

Lorsque le serveur Web 4D reçoit un formulaire posté, il appelle automatiquement la méthode projet nommée COMPILER_WEB (si elle existe). Cette méthode doit contenir toutes les directives de typage et/ou d'initialisation des variables, en l'occurrence les variables dont les noms sont ceux des champs du formulaire Web. Elle sera utilisée par le compilateur en cas de compilation de la base.
La méthode COMPILER_WEB est commune à tous les formulaires Web. Par défaut, la méthode COMPILER_WEB n'existe pas. Vous devez la créer explicitement.

Note : Alternativement, vous pouvez utiliser la commande LIRE VARIABLES FORMULAIRE WEB, permettant de récupérer le nom et la valeur de tous les champs inclus dans une page HTML soumise au serveur.

Web Services : La méthode projet COMPILER_WEB est également appelée, si elle existe, à chaque requête SOAP acceptée. Vous devez utiliser cette méthode pour déclarer toutes les variables 4D associées à des arguments SOAP entrants et ce, pour toutes les méthodes publiées comme Web Services. En effet, l'utilisation de variables process dans les méthodes Web Services nécessite leur déclaration avant l'appel de la méthode. Pour plus d'informations sur ce point, reportez-vous à la description de la commande DECLARATION SOAP.

Server-side Image Mapping — Gérer des images interactives

Comme cela est décrit dans la section Utiliser le mode contextuel, quand un formulaire 4D est utilisé comme page Web, 4D peut fournir une image interactive côté serveur ("Server-side Image Mapping") au moyen de boutons invisibles placés par-dessus une image statique.

Si vous envoyez un document HTML avec ENVOYER FICHIER HTML ou ENVOYER BLOB HTML, vous pouvez aussi lier une variable 4D avec des objets HTML Image interactive (INPUT TYPE="IMAGE"). Par exemple, vous créez un objet HTML Image interactive dont le nom est bImageMap (vous prenez le nom que vous voulez). Chaque fois que vous cliquez sur l'image côté navigateur, un submit avec la position du clic est renvoyé au Serveur Web 4D. Pour récupérer les coordonnées de ce clic (exprimé par rapport au coin supérieur gauche de l'image), il suffit de lire la valeur des variables process 4D bImageMap_X et bImageMap_Y (de type Entier long) qui contiennent les coordonnées horizontales et verticales du clic. Ces variables doivent impérativement être initialisées dans la méthode projet COMPILER_WEB (cf. paragraphe précédent).

Dans la page HTML, vous écrivez, par exemple :

```
<P><INPUT TYPE="image" SRC="MonImage.GIF" NAME="bImageMap" BORDER=0></P>
```

La méthode 4D qui envoie la page HTML contient simplement :

```
ENVOYER FICHIER HTML("CettePage.HTM")
```

Dans la méthode projet COMPILER_WEB, vous écrivez :

```
C_ENTIER LONG(bImageMap_X;bImageMap_Y)
bImageMap_X:=-1    `Initialisation de la variable
bImageMap_Y:=-1    `Initialisation de la variable
```

Ensuite, dans la méthode 4D appelée par l'action POST du formulaire — ou dans la méthode courante une fois que la méthode d'action POST a émis un ENVOYER FICHIER HTML ("") — vous récupérez les coordonnées du clic dans les variables bImageMap_X et bImageMap_Y :

```
Si (($bImageMap_X#-1)&($bImageMap_Y#-1))
    ` Faire quelque chose en fonction de ces coordonnées
Fin de si
```

Encapsuler du JavaScript

Le code source JavaScript encapsulé dans les documents HTML est supporté par le serveur Web 4D, ainsi que l'insertion de fichiers JavaScript .js dans des documents HTML (par exemple <SCRIPT SRC="...").

Avec ENVOYER FICHIER HTML ou ENVOYER BLOB HTML en mode standard, vous envoyez une page que vous avez préparée avec un éditeur HTML ou construite avec 4D et sauvegardée comme document sur disque. Dans les deux cas, vous avez tout contrôle sur la page et, par exemple, vous pouvez insérer des scripts JavaScript dans la section HEAD du document et utiliser des scripts avec une balise FORM. Ainsi, dans l'exemple ci-dessus, le script renvoie le formulaire "frm" car vous avez donné un nom au formulaire. Vous pouvez également déclencher, accepter ou rejeter la soumission du formulaire au niveau de la balise FORM.

En mode contextuel, si vous encapsulez de l'HTML dans un formulaire 4D, vous n'avez pas de contrôle sur la section HEAD (en-tête) ni sur la déclaration du FORM (formulaire). L'aire d'action des scripts est donc différente. Ainsi, vous ne pouvez pas accéder au formulaire HTML par son nom. Cependant, comparez la fonction JavaScript Is4DWebServer de l'exemple ci-dessus avec celui-ci :

```
function Is4DWebServer(){
   return (document.forms[0].vs4D.value=="4D4D")
}
```

Les deux fonctions font la même chose. La seconde utilise la propriété forms de l'objet HTML document pour accéder à l'objet par l'élément forms[0]. Le résultat est que tout fonctionne, même si vous ne connaissez pas le nom du formulaire que 4D a ou n'a pas donné à la page HTML traduite.

Note : 4D supporte le transport d'Applets Java.

Référence

Balises HTML 4D, ENVOYER BLOB HTML, ENVOYER FICHIER HTML, URLs et actions de formulaires.

Le serveur Web 4D propose plusieurs URLs et actions de formulaires HTML spéciaux vous permettant d'effectuer différentes actions dans votre base de données, en mode contextuel ou en mode sans contexte.

Ces URLs sont les suivants :
• 4DMETHOD/, permettant de lier un objet HTML à une méthode projet de votre base en mode contextuel,
• 4DACTION/, permettant de lier un objet HTML à une méthode projet de votre base en mode sans contexte,
• 4DCGI/, permettant d'appeler la Méthode base Sur connexion Web depuis tout objet HTML.

En outre, le serveur Web 4D accepte plusieurs URLs supplémentaires :
• /4DSTATS, /4DHTMLSTATS, /4DCACHECLEAR et /4DWEBTEST, vous permettent d'obtenir diverses informations sur le fonctionnement de votre site Web 4D. Ces URLs sont décrits dans la section Informations sur le site Web.
• /4DWSDL, permettant d'accéder au fichier de déclaration des Web Services publiés sur le serveur. Pour plus d'informations, reportez-vous à la section Commandes du thème Web Services (Serveur) et au manuel *Mode Développement*.

URL 4DACTION/

Syntaxe : 4DACTION/MaMéthode{/Param}
Mode : Sans contexte. Appelé depuis le mode contextuel, provoque la fermeture du process Web et le passage en mode sans contexte.
Utilisation : URL ou Action de formulaire.

Cet URL vous permet de lier un objet HTML (texte, image, bouton...) à une méthode projet 4D en mode sans contexte. Le lien sera du type /4DACTION/MAMETH/PARAMS où MAMETH est le nom de la méthode projet 4D à exécuter lorsque l'utilisateur clique sur le lien et PARAMS un paramètre optionnel de type Texte passé à la méthode dans $1 (reportez-vous ci-dessous au paragraphe “Les paramètres Texte passés aux méthodes via des URLs”).
Lorsque 4D reçoit une requête /4DACTION/MAMETH/PARAMS, la Méthode base Sur authentification Web (si elle existe) est appelée. Si elle retourne Vrai, la méthode MAMETH est exécutée.

4DACTION/ peut être associé à un URL dans une page Web statique. La syntaxe de l’URL sera de la forme suivante :

```
<A HREF="/4DACTION/MAMETH/PARAMS">Faire Quelque Chose</A>
```

La méthode projet MAMETH doit généralement retourner une “réponse” (envoi de page HTML via ENVOYER FICHIER HTML ou ENVOYER BLOB HTML, etc.). Veillez à effectuer les traitements les plus courts possibles, afin de ne pas bloquer le navigateur.

Note : Une méthode appelée par 4DACTION ne doit pas faire appel à des éléments d’interface (DIALOGUE, ALERTE...).

Attention : Pour qu’une méthode 4D puisse être exécutée via l’URL 4DACTION/, elle doit disposer de l'attribut “Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT” (désélectionné par défaut), défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section Sécurité des connexions.

Exemple

Cet exemple décrit l'association de l'URL 4DACTION/ à un objet HTML image afin d'afficher dynamiquement une image dans la page. Vous insérez dans une page HTML statique les instructions suivantes :
<IMG SRC="/4DACTION/PICTFROMLIB/1000">

Le code de *PICTFROMLIB* est le suivant :

```
C_TEXTE($1) `Ce paramètre doit toujours être déclaré
C_IMAGE($VarImage)
C_BLOB($VarBlob)
C_ENTIER LONG($Numéro)
   `On récupère le numéro d’image dans la chaîne $1
$Numéro:=Num(Sous chaine($1;2;99))
LIRE IMAGE DANS BIBLIOTHEQUE($Numéro;$VarImage)
IMAGE VERS GIF($VarImage;$VarBlob)
ENVOYER BLOB HTML ($VarBlob;"image/gif")
```

4DACTION pour poster des formulaires

Le serveur Web 4D permet une facilité supplémentaire lorsque vous souhaitez utiliser des formulaires “postés”, c’est-à-dire des pages HTML statiques renvoyant des données au serveur Web. L’action du formulaire doit impérativement débuter par /4DACTION/NomMéthode.

Note : Un formulaire peut être soumis dans deux modes (4D accepte les deux) :
• POST, généralement utilisé pour envoyer des données vers le serveur Web - dans une base de données.
• GET, généralement utilisé pour interroger le serveur Web - données en provenance de la base.

Lorsque le serveur Web reçoit un formulaire posté, il appelle la méthode projet COMPILER_WEB (si elle existe), puis la Méthode base Sur authentification Web (si elle existe). Si elle retourne Vrai, la méthode NomMéthode est exécutée. 4D analyse les champs HTML présents dans le formulaire, récupère leurs valeurs et remplit automatiquement des variables 4D avec leur contenu. Il suffit pour cela que les champs du formulaire et les variables 4D portent le même nom.

Vous pouvez également utiliser la commande LIRE VARIABLES FORMULAIRE WEB, permettant de récupérer le nom et la valeur de tous les champs inclus dans une page HTML soumise au serveur.

Note : Pour plus d'informations, reportez-vous à la section Associer des objets 4D à des objets HTML.

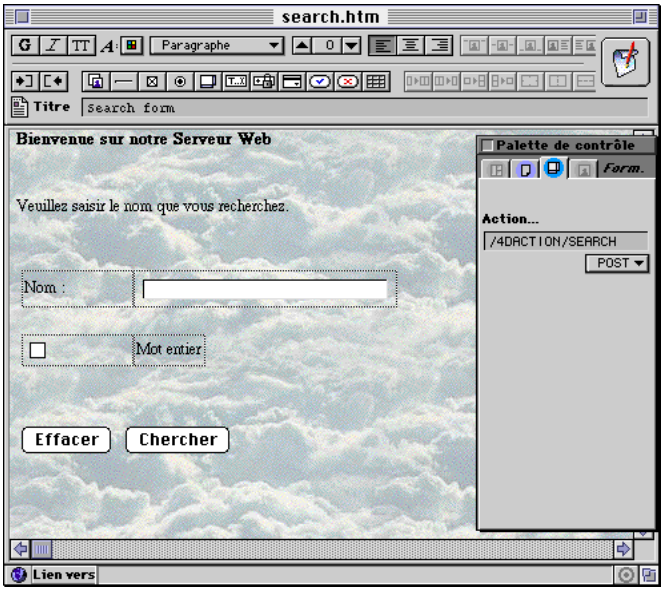
La syntaxe HTML à appliquer dans le formulaire est du type suivant :
• pour la définition de l'action du formulaire :
<FORM ACTION="/4DACTION/NomMéthode" METHOD=POST>
• pour la définition d'un champ du formulaire :
<INPUT TYPE=Type de champ NAME=nom du champ VALUE="Valeur par défaut">

Pour chaque champ du formulaire, 4D affecte la valeur du champ à la variable de même nom.
Pour les options de formulaires (par exemple les cases à cocher), 4D affecte la valeur 1 à la variable associée s'il est coché, sinon 0.
Pour les saisies de type numérique, 4D effectue une conversion Alpha–>Numérique de la valeur du champ.

Note : Si un champ du formulaire est nommé OK (par exemple un bouton Submit), la variable système *OK* de 4D est mise à 1 si la valeur du champ est non vide, sinon 0.

Exemple

Dans une base Web 4D démarrée et utilisée en mode “sans contexte”, nous souhaitons que les navigateurs puissent effectuer des recherches parmi les enregistrements par l’intermédiaire d’une page HTML statique. Cette page s’intitule “search.htm”. La base contient d’autres pages statiques, permettant par exemple d’afficher le résultat de la recherche (“results.htm”). Le type POST a été associé à la page, ainsi que l’action /4DACTION/SEARCH. Voici la page telle qu’elle apparaît dans un éditeur HTML, ici Adobe® PageMill™ :



Voici le code HTML correspondant à cette page :

```
<FORM ACTION="/4DACTION/PROCESSFORM" METHOD=POST>
<INPUT TYPE=TEXT NAME=VNAME VALUE=""><BR>
<!— Habituellement on met le nom du bouton dans VALUE, pour des raisons d’interprétation il faut mettre un nombre dans VALUE—>
<INPUT TYPE=CHECKBOX NAME=EXACT VALUE="1">Mot entier<BR>
<!— OK est un cas particulier—>
<INPUT TYPE=SUBMIT NAME=OK VALUE="Chercher">
</FORM>
```

En cours d'utilisation, vous tapez "ABCD" dans la zone de saisie, vous cochez l'option et validez en cliquant sur **Chercher**.
4D appelle alors la méthode projet COMPILER_WEB. Voici son contenu :

```
C_TEXTE(VNAME)
VNAME:=""
C_ENTIER LONG(vEXACT)
vEXACT:=0
OK:=0 `cas particulier
```

Dans l'exemple, *VNAME* contient la chaîne "ABCD", *vEXACT* a pour valeur 1, et *OK* a pour valeur 1 (car on a cliqué sur un bouton dont le nom est OK).
4D appelle la Méthode base Sur authentification Web (si elle existe), puis la méthode projet *PROCESSFORM*, dont voici le contenu :

```
Si (OK=1)
    Si (vEXACT=0) `Si l'option n'a pas été cochée
        vNAME:=VNAME+"@"
    Fin de si
    CHERCHER([Jockeys];[Jockeys]Nom=vNAME)
    vLIST:=Caractere(1)   `Retourne la liste en HTML
    DEBUT SELECTION([Jockeys])
    Tant que (Non(Fin de selection([Jockeys])))
        vLIST:=vLIST+[Jockeys]Nom+" "+[Jockeys]Tél+"<BR>"
        ENREGISTREMENT SUIVANT([Jockeys])
    Fin tant que
    ENVOYER FICHIER HTML("results.htm")   `Envoi de la liste dans le
                `formulaire results.htm, qui contient une référence à la variable vLIST,
                `par exemple <!—4DVAR vLIST—>
    ...
Fin de si
```

URL 4DMETHOD/

Syntaxe : 4DMETHOD/MaMéthode{/Param}
Mode : Contextuel. Appelé depuis le mode sans contexte, provoque le passage en mode contextuel.
Utilisation : URL ou Action de formulaire.

Cet URL vous permet de lier un objet HTML (texte, bouton...) à une méthode projet 4D en mode contextuel. Le lien sera du type /4DMETHOD/Nom_Méthode/Param où Nom_Méthode est le nom de la méthode projet 4D à exécuter lorsque l'utilisateur clique sur le lien et Param un paramètre optionnel de type Texte passé à la méthode dans $1 (reportez-vous ci-dessous au paragraphe "Les paramètres Texte passés aux méthodes via des URLs").

L'envoi d'une requête HTTP débutant par l'URL 4DACTION/ provoque l'exécution de la méthode projet 4D désignée. La méthode projet peut elle-même afficher des formulaires 4D, d'autres pages HTML, etc.

Lorsque 4D reçoit une requête /4DMETHOD, la Méthode base Sur authentification Web (si elle existe) est appelée. Si elle retourne Vrai, la Méthode base Sur connexion Web (si elle existe) est appelée, puis la méthode Nom_Méthode est exécutée avec comme paramètre (dans $1) la chaîne /Param.

Si vous affectez /4DMETHOD/Nom_Méthode en tant qu'action de formulaire à une page HTML statique, la méthode est exécutée lorsque l'utilisateur clique sur un bouton **Submit** dans le formulaire HTML. Reportez-vous à l'exemple de la commande ENVOYER FICHIER HTML.
Lorsque vous intégrez des pages HTML dans 4D, vous utilisez généralement des boutons de type Normal ou Submit.
Pour définir l'action d'un formulaire, la syntaxe HTML à employer est de la forme suivante :
<FORM ACTION="/4DMETHOD/NomMéthode" METHOD=POST>
Pour plus d'informations sur les formulaires postés, reportez-vous au paragraphe précédent.

Attention : Pour qu'une méthode 4D puisse être exécutée via l'URL 4DMETHOD/, elle doit disposer de l'attribut "Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT" (désélectionné par défaut), défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section Sécurité des connexions.

URL 4DCGI/

Syntaxe : 4DCGI/<action>
Mode : Tous.
Utilisation : URL.

Lorsque le serveur Web 4D reçoit l'URL /4DCGI/<action>, la Méthode base Sur authentification Web (si elle existe) est appelée. Si elle retourne Vrai, le serveur Web appelle la Méthode base Sur connexion Web en passant l'URL "tel quel" dans $1.

L'URL 4DCGI/ ne correspond à aucun fichier. Il a pour unique rôle d'appeler 4D via la Méthode base Sur connexion Web. Le paramètre "<action>" peut contenir n'importe quel type d'information.

Cet URL vous permet donc d'effectuer tout type de traitement. Il vous suffit de tester la valeur de $1 dans la Méthode base Sur connexion Web ou dans une de ses sous-méthodes et d'effectuer dans 4D le traitement adéquat. Par exemple, vous pouvez construire des pages HTML statiques entièrement personnalisées d'ajout, de recherche ou de tri d'enregistrements, ou encore générer des images GIF à la volée. Des exemples d'utilisation de cet URL sont fournis dans les descriptions des commandes IMAGE VERS GIF et ENVOYER REDIRECTION HTTP.

A l’issue du traitement, une “réponse” doit être retournée, à l’aide d’une des commandes d’envoi de données (ENVOYER FICHIER HTML, ENVOYER BLOB HTML, etc.).

ATTENTION : Veillez à effectuer les traitements les plus courts possibles, afin de ne pas bloquer le navigateur.

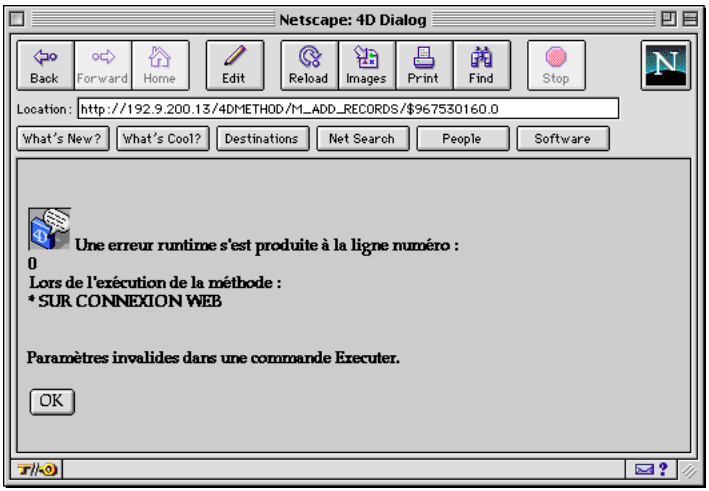
Paramètres texte passés aux méthodes 4D appelées via des URLs

4D envoie des paramètres Texte aux méthodes 4D appelées par des URLs spéciaux (4DMETHOD/, 4DACTION/ et 4DCGI/), en mode contextuel et en mode sans contexte. Voici quelques remarques sur ces paramètres :

• Même si vous n'utilisez pas ces paramètres, vous devez les déclarer explicitement avec la commande C_TEXTE, sinon des erreurs runtime se produiront lorsque vous accèderez par le Web à une base exécutée en mode compilé.
• Le paramètre $1 retourne des données supplémentaires placées à la fin de l'URL, et peut être utilisé pour passer des données de l'environnement HTML vers l'environnement 4D.

Erreurs runtime en mode compilé

Prenons un exemple : vous exécutez une méthode liée à un objet HTML. Vous obtenez l'écran suivant dans votre navigateur Web :



Cette erreur runtime est provoquée par l'absence de déclaration du paramètre texte $1 dans la méthode 4D appelée lorsque vous avez cliqué sur le lien HTML. Comme le contexte de l'exécution est la page HTML courante, l'erreur fait référence à la “ligne 0” de la méthode qui a envoyé la page au navigateur Web.

Pour reprendre l'exemple de la section Premiers pas, le paramètre texte $1 est déclaré explicitement dans les méthodes M_ADD_RECORDS et M_LIST_RECORDS :

```
    `Méthode projet M_ADD_RECORDS
C_TEXTE($1) ` Ce paramètre DOIT être explicitement déclaré
Repeter
    AJOUTER ENREGISTREMENT ([Clients])
Jusque (OK=0)

    `Méthode projet M_LIST_RECORDS
C_TEXTE($1) ` Ce paramètre DOIT être explicitement déclaré
TOUT SELECTIONNER ([Clients])
MODIFIER SELECTION([Clients])
```

Lorsque que ces déclarations sont effectuées, les erreurs runtime en mode compilé ne se produisent pas.

Paramètres à déclarer explicitement dans la méthode appelée

Vous devez déclarer différents paramètres en fonction de l'origine et de la nature de l'appel de la méthode 4D.

• Méthode base Sur authentification Web (si elle existe) et Méthode base Sur connexion Web
Vous devez déclarer les six paramètres de la connexion :

```
    ` Méthode base Sur connexion Web
C_TEXTE($1;$2;$3;$4;$5;$6) ` Ces paramètres DOIVENT être explicitement déclarés
```

• Méthode appelée par l'URL 4DMETHOD/
Vous devez déclarer le paramètre $1 :

```
    ` Méthode appelée par l'URL 4DMETHOD/
C_TEXTE($1) ` Ce paramètre DOIT être explicitement déclaré
```

• Méthode appelée par l'URL 4DACTION/
Vous devez déclarer le paramètre $1 :

```
    ` Méthode appelée par l'URL 4DACTION/
C_TEXTE($1) ` Ce paramètre DOIT être explicitement déclaré
```

• Méthode appelée par la balise 4DSCRIPT/
La méthode retourne une valeur dans $0. Vous devez déclarer les paramètres $0 et $1 :

```
    ` Méthode appelée par 4DSCRIPT/ sous forme de commentaire HTML
C_TEXTE($0; $1) ` Ces paramètres DOIVENT être explicitement déclarés
```

Référence

Associer des objets 4D à des objets HTML, LIRE VARIABLES FORMULAIRE WEB, Premiers pas, Utiliser le mode contextuel.

Le serveur Web de 4D fournit diverses balises HTML spécifiques à 4D, permettant d'insérer des références à des expressions ou variables 4D, ou différents types de traitements, à l'intérieur des pages HTML envoyées par le serveur Web 4D, par exemple à l'aide des commandes ENVOYER FICHIER HTML et ENVOYER BLOB HTML. Ces pages sont appelées **pages semi-dynamiques**.

Ces balises doivent être insérées sous forme de commentaires HTML (<!--#LaBalise LeContenu--> en code HTML). La plupart des éditeurs HTML proposent des fonctions facilitant l'insertion de commentaires dans les pages statiques.

Les balises HTML 4D disponibles sont les suivantes :
- 4DVAR, pour insérer des variables et expressions 4D
- 4DHTMLVAR, semblable à 4DVAR mais insérant du code HTML
- 4DSCRIPT, pour exécuter une méthode 4D
- 4DINCLUDE, pour insérer une page HTML dans une autre page
- 4DIF, 4DELSE et 4DENDIF, pour insérer des conditions dans le code HTML
- 4DLOOP et 4DENDLOOP, pour insérer des boucles dans le code HTML

Note de compatibilité : Dans les versions 6.0.x de 4D, l'insertion de variables 4D dans les pages statiques s'effectuait à l'aide d'une notation utilisant des crochets ([NomVar]). Si vous travaillez avec une base créée avec une version précédente de 4D et souhaitez utiliser la notation standard <!--#4DVAR NomVar-->, assurez-vous que l'option "Utiliser les commentaires 4DVAR au lieu des crochets" dans la boîte de dialogue des Préférences est bien cochée (reportez-vous à la section Paramétrages du serveur Web).

Traitement des balises HTML 4D

L'analyse du contenu des pages semi-dynamiques envoyées par 4D s'effectue au moment de l'appel à ENVOYER FICHIER HTML (.htm, .html, .shtm, .shtml), ENVOYER BLOB HTML (blob de type text/html) et lors de l'envoi de pages appelées via des URLs.
Toutefois en mode sans contexte, à des fins d'optimisation, les pages suffixées ".htm" et ".html" ne sont PAS analysées. Pour "forcer" l'analyse des pages HTML dans ce cas, vous devez les suffixer ".shtm" ou ".shtml" (par exemple http://www.server.com/dir/page.shtm). Un exemple d'utilisation de ce type de page est fourni dans la description de la routine STATISTIQUES DU CACHE WEB.

Voici un tableau récapitulatif des cas dans lesquels 4D analyse les balises contenues dans les pages HTML envoyées aux navigateurs Web :

Conditions d'envoi	**Analyse du contenu des pages envoyées**	
	Mode contextuel	**Mode sans contexte**
• Extensions des pages (cas général) :		
.htm, .html, .shtm, .shtml (pages HTML)	X	X
.xml, .xsl (pages XML)	X	X
.wml (pages WML)	X	X
• Pages appelées par des URLs	X	X, sauf pages suffixées ".htm" ou ".html"
• Commande ENVOYER FICHIER HTML	X	X
• Commande ENVOYER BLOB HTML (si le BLOB est du type "text/html")	X	X
• Inclusion par la balise <!--#4DINCLUDE-->	X	X
• Inclusion par la balise {mapage.htm}	X	-

En vue d'une exploitation par 4D, un commentaire HTML doit toujours être de la forme <!--#4D...-->. Attention, certains éditeurs HTML ajoutent automatiquement d'autres informations au sein d'un commentaire, ce qui peut entraîner des erreurs d'analyse.
La présence d'autres commentaires HTML (par exemple <!--Début de liste-->) est toutefois possible.

Si un commentaire <!--#4D... ne se termine pas par -->, un texte de la forme "<!--#4D... : --> attendu" sera inséré et l'analyse sera interrompue à ce niveau (la page sera tout de même envoyée pour indiquer l'erreur).

Il est possible d'imbriquer les différents types de commentaires. Voici par exemple une structure HTML parfaitement envisageable :

```
<HTML>
...
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->                                  (Appel de méthode)
<!--#4DIF (mavar=1)-->                                        (Si condition)
        <!--#4DINCLUDE banner1.html-->                        (Insertion d'une sous page)
<!--#4DENDIF-->                                               (Fin de si)
<!--#4DIF (mavar=2)-->
        <!--#4DINCLUDE banner2.html-->
<!--#4DENDIF-->
<!--#4DLOOP [TABLE]-->                                        (Boucle sur la sélection courante)
<!--#4DIF ([TABLE]ValNum>10)-->                               (Si [TABLE]ValNum>10)
        <!--#4DINCLUDE subpage.html-->                        (Inclusion d'une sous page)
<!--#4DELSE-->                                                (Sinon)
        <B>Valeur : <!--#4DVAR [TABLE]ValNum--></B><BR>       (Affichage d'un champ)
<!--#4DENDIF-->
<!--#4DENDLOOP-->                                               (Fin de boucle)
</BODY>
</HTML>
```

Balise 4DVAR

Syntaxe : <!--#4DVAR NomVar--> ou <!--#4DVAR Expression4D-->

La balise <!--#4DVAR NomVar--> vous permet d'insérer une référence à une variable ou à une expression 4D à tout endroit d'une page HTML. Par exemple, si vous écrivez :
<P>Bienvenue sur <!--#4DVAR vtNomSite-->!</P>
La valeur de la variable 4D vtNomSite sera insérée dans la page HTML lors de son envoi.

Vous pouvez également insérer du code HTML dans une variable texte 4D. Pour cela, il vous suffit de faire débuter la variable par le code 1 (par exemple, vtHTML:=Caractere(1)+"...code HTML..."). Un autre moyen consiste à utiliser la balise 4DHTMLVAR.

Il est aussi possible d'insérer des *expressions* 4D dans les commentaires HTML 4D à l'aide de la balise 4DVAR. Vous pouvez par exemple insérer directement le contenu d'un champ (<!--#4DVAR [nomTable]nomChamp-->), un élément de tableau (<!--#4DVAR tab{1}-->) ou une méthode retournant une valeur (<!--#4DVAR mamethode-->).
La conversion de l'expression obéit aux mêmes règles que la conversion d'une variable. En outre, l'expression doit satisfaire aux règles de syntaxe de 4D.

Note : L'exécution d'une méthode 4D avec 4DVAR est soumise à la valeur de l'attribut "Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section Sécurité des connexions.

Bien qu'une expression puisse contenir directement des appels de fonctions 4D, il est déconseillé d'utiliser ce mode de fonctionnement, pour des raisons de localisation. Par exemple le commentaire <!--#4DVAR Date du jour-->, bien que correctement interprété avec un 4D français, ne sera pas traité avec un 4D anglais. Il en va de même pour le traitement des nombres réels (car le séparateur décimal peut varier d'un pays à l'autre). Dans ces deux cas, il est préférable d'affecter une variable par programmation.

En cas d'erreur d'évaluation, le texte inséré sera de la forme "<!--#4DVAR mavar--> : ## erreur # code d'erreur".

Notes :
• Vous devez utiliser des variables process.
• Il est possible d'afficher le contenu d'un champ image. En outre (en mode contextuel uniquement), vous pouvez également afficher le contenu d'une variable image. En revanche, dans les deux modes, il n'est pas possible d'afficher le contenu d'un élément de tableau image.

• Comme le HTML est orienté texte, vous travaillerez généralement avec des variables Texte. Toutefois, vous pouvez également utiliser des variables BLOB (ce qui vous permet de vous affranchir de la limite des 32 000 caractères inhérente aux variables de type texte). Il vous suffit pour cela de générer des BLOBs en mode Texte sans longueur.
• Des exemples d'utilisation de la balise 4DVAR sont fournis dans la section Associer des objets 4D à des objets HTML.

Balise 4DHTMLVAR

Syntaxe : <!--#4DHTMLVAR NomVar--> ou <!--#4DHTMLVAR Expression4D-->

Cette balise permet d'évaluer une variable ou une expression 4D et de l'insérer dans une page en tant qu'expression HTML. En fait, cette balise produit exactement les mêmes effets que la balise <!--#4DVAR mavar-->, lorsque *mavar* débute par le caractère de code 1.

Par exemple, voici les résultats de l'insertion de la variable Texte 4D *mavar* à l'aide des balises disponibles :

Valeur de *mavar*	**Balises**	**Insertion dans la page Web**
mavar:="<B>"	<!--#4DVAR mavar-->	<B>
mavar:=Caractere(1)+"<B>"	<!--#4DVAR mavar-->	<B>
mavar:="<B>"	<!--#4DHTMLVAR mavar-->	<B>

En cas d'erreur d'évaluation, le texte inséré sera de la forme "<!--#4DHTMLVAR mavar--> : ## erreur # code d'erreur".

Note : L'exécution d'une méthode 4D avec 4DHTMLVAR est soumise à la valeur de l'attribut "Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section Sécurité des connexions.

Note : La variable doit être exprimée à l'aide de la table de caractères ISO Latin-1 (pour plus d'informations, reportez-vous à la description de la commande Mac vers ISO).

Balise 4DSCRIPT/

Syntaxe : <!--#4DSCRIPT/NomMéthode/Param-->

La balise 4DSCRIPT/ vous permet d'exécuter des méthodes 4D au moment de l'envoi de pages HTML statiques. La présence dans une page statique du commentaire HTML <!--#4DSCRIPT/NomMéthode/Param--> provoque l'exécution de la méthode NomMéthode avec le paramètre *Param* sous forme de chaîne dans $1. Au chargement de la page, 4D appelle la Méthode base Sur authentification Web (si elle existe). Si elle retourne Vrai, 4D exécute la méthode.

La méthode retourne alors du texte dans $0. Si la chaîne débute par le caractère de code 1, elle est considérée comme du HTML (même principe que pour les variables).

Note : L'exécution d'une méthode avec 4DSCRIPT est soumise à la valeur de l'attribut "Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT" défini dans les propriétés de la méthode. Pour plus d'informations sur ce point, reportez-vous à la section Sécurité des connexions.

Note : En mode contextuel, la méthode est exécutée dans le contexte.

Par exemple, vous insérez dans une page statique le commentaire "Nous sommes le <!--#4DSCRIPT/MAMETH/MONPARAM-->". Au chargement de la page, 4D appelle la Méthode base Sur authentification Web (si elle existe) puis la méthode MAMETH en lui passant comme paramètre (dans $1) la chaîne "/MONPARAM".
La méthode retourne du texte dans $0 (par exemple "28/10/03"), l'expression "Nous sommes le <!--#4DSCRIPT/MAMETH/MONPARAM-->" devient alors "Nous sommes le 28/10/03".

Le code de *MAMETH* est :

```
C_TEXTE($0;$1) `Ces paramètres doivent TOUJOURS être déclarés
$0:=Chaine(Date du jour)
```

Note : Une méthode appelée par 4DSCRIPT ne doit pas faire appel à des éléments d'interface (DIALOGUE, ALERTE...)

Comme 4D exécute les méthodes dans leur ordre d'apparition, il est tout à fait possible de faire appel à une méthode qui fixe la valeur de plusieurs variables elles-mêmes référencées plus loin dans le document, quel que soit le mode utilisé.

Note : Vous pouvez insérer autant de commentaires <!--#4DSCRIPT...--> que vous voulez dans une page statique.

Note de compatibilité : Dans les versions précédentes de 4D, la même balise, 4DACTION, pouvait être utilisée soit sous forme d'URL (par exemple http://monserver/4DACTION/meth), soit sous forme de commentaire HTML dans une page statique (<!--#4DACTION/meth-->). Cette diversité d'utilisations pouvant entraîner des confusions, la balise de commentaire 4DSCRIPT permet désormais de dissocier les deux usages. La balise 4DSCRIPT s'utilise exclusivement sous forme de commentaire HTML (<!--#4DSCRIPT/meth-->) et produit exactement les mêmes effets que <!--#4DACTION/meth-->. 4DACTION est désormais réservée à une utilisation en tant qu'URL.

Balise 4DINCLUDE

Syntaxe : <!--#4DINCLUDE chemin-->

Ce commentaire permet d'inclure, dans une page HTML, le corps d'une autre page HTML (désignée par le paramètre chemin). Le corps d'une page HTML désigne ce qui est compris entre les balises <BODY> et </BODY> (les balises elles-mêmes ne sont pas incluses).
Le commentaire <!--#4DINCLUDE --> s'avère particulièrement utile en combinaison avec les tests (<!--#4DIF-->) ou les boucles (<!--#4DLOOP-->). Il est également pratique pour inclure des bannières en fonction d'un critère, ou de manière aléatoire.
Au moment de l'inclusion, quels que soient le mode et l'extension du nom du fichier, 4D analyse la page appelée puis insère son contenu — éventuellement modifié — dans la page d'où provient l'appel 4DINCLUDE.

Le placement dans le cache Web d'une page incluse à l'aide du commentaire <!--#4DINCLUDE --> répond aux mêmes règles que celles des pages demandées via un URL ou envoyées par la commande ENVOYER FICHIER HTML.

Passez dans chemin le chemin d'accès au document à inclure. **Attention** : Dans le cas de la balise 4DINCLUDE, le chemin d'accès doit être défini relativement au document en cours d'analyse, c'est-à-dire au document "parent", et non à la racine du serveur. Utilisez la barre oblique (/) comme séparateur de dossiers et les deux-points (..) pour remonter d'un niveau hiérarchique (syntaxe HTML).

Note : Lorsque vous utilisez la balise 4DINCLUDE avec la commande TRAITER BALISES HTML, le dossier par défaut est le dossier contenant le fichier de structure de la base.

Le nombre de <!--#4DINCLUDE chemin--> au sein d'une page n'est pas limité. Toutefois, les appels à <!--#4DINCLUDE chemin--> ne peuvent s'effectuer que sur 1 niveau. Cela signifie que par exemple vous ne pouvez pas insérer le commentaire <!--#4DINCLUDE mondoc3.html--> dans le corps de la page mondoc2.html, elle-même appelée par <!--#4DINCLUDE mondoc2--> inséré dans mondoc1.html.
En outre, 4D contrôle que les inclusions ne sont pas récursives.

En cas d'erreur, le texte inséré est de la forme "<!--#4DINCLUDE chemin--> : Le document ne peut pas être ouvert".

Note : En mode contextuel, si une page est insérée dans un formulaire via un marqueur {mapage.html} inséré dans une zone de texte statique, les éventuels commentaires 4DINCLUDE qu'elle contient seront ignorés.

Exemples

<!--#4DINCLUDE souspage.html-->
<!--#4DINCLUDE dossier/souspage.html-->
<!--#4DINCLUDE ../dossier/souspage.html-->

Balises 4DIF, 4DELSE et 4DENDIF

Syntaxe : <!--#4DIF expression--> <!--#4DELSE--> <!--#4DENDIF-->

Utilisé en conjonction avec les commentaires <!--#4DELSE--> (optionnel) et <!--#4DENDIF-->, le commentaire <!--#4DIF expression--> offre la possibilité d'exécuter du code HTML de manière conditionnelle.
Le paramètre expression peut contenir toute expression 4D valide retournant une valeur booléenne. Elle doit figurer entre parenthèses et respecter les règles de syntaxe de 4D.

Les blocs <!--#4DIF expression--> ... <!--#4DENDIF--> peuvent être imbriqués sur plusieurs niveaux. Comme dans 4D, chaque <!--#4DIF expression--> doit avoir un <!--#4DENDIF--> correspondant.
En cas d'erreur d'évaluation, le texte "<!--#4DIF expression--> : Une expression booléenne était attendue" est inséré à la place du contenu situé entre <!--#4DIF --> et <!--#4DENDIF-->. De même, s'il n'y a pas autant de <!--#4DENDIF--> que de <!--#4DIF -->, le texte "<!--#4DIF expression--> : 4DENDIF attendu" est inséré à la place du contenu situé entre <!--#4DIF --> et <!--#4DENDIF-->.

Exemple

Cet exemple de code inséré dans une page HTML statique affiche un libellé différent en fonction du résultat de l'expression vnom#"" :

```
<BODY>
...
<!--#4DIF (vnom#"")-->
Noms commençant par <!--#4DVAR vnom-->.
<!--#4DELSE-->
Aucun nom n'a été trouvé.
<!--#4DENDIF-->
...
</BODY>
```

Balises 4DLOOP et 4ENDLOOP

Syntaxe : <!--#4DLOOP condition--> <!--#4DENDLOOP-->

Ce commentaire permet de répéter une portion de code HTML tant que la condition est remplie. La portion est délimitée par <!--#4DLOOP--> et <!--#4DENDLOOP-->.
Les blocs <!--#4DLOOP condition--> ... <!--#4DENDLOOP--> peuvent être imbriqués. Comme dans 4D, chaque <!--#4DLOOP condition--> doit avoir un <!--#4DENDLOOP--> correspondant.

Il existe trois types de conditions :

• **<!--#4DLOOP [table]-->**
Cette syntaxe effectue une boucle pour chaque enregistrement de la sélection courante de table dans le process en cours. La portion de code HTML située entre les deux commentaires est répétée pour chaque enregistrement de la sélection courante.

Note : Lors de l'utilisation de 4DLOOP avec une table, les enregistrements sont chargés en mode Lecture seule.

L'exemple de code HTML suivant :

```
<!--#4DLOOP [Personnes]-->
<!--#4DVAR [Personnes]Nom--> <!--#4DVAR [Personnes]Prenom--><BR>
<!--#4DENDLOOP-->
```

... pourrait se traduire en langage 4D par :

```
DEBUT SELECTION([Personnes])
Tant que(Non(Fin de selection([Personnes])))
    ...
    ENREGISTREMENT SUIVANT([Personnes])
Fin tant que
```

• **<!--#4DLOOP tableau-->**
Cette syntaxe effectue une boucle pour chaque élément du tableau. L'indice courant du tableau est incrémenté à chaque répétition de la portion de code HTML.
Il n'est pas possible d'utiliser cette syntaxe avec des tableaux à deux dimensions. Dans ce cas, la solution consiste à combiner une méthode et des boucles imbriquées.

L'exemple de code HTML suivant :

```
<!--#4DLOOP tab_noms-->
<!--#4DVAR tab_noms{tab_noms}--><BR>
<!--#4DENDLOOP-->
```

... pourrait se traduire en langage 4D par :

```
Boucle($indice;1;Taille tableau(tab_noms))
    tab_noms:=$indice
    ...
Fin de boucle
```

• **<!--#4DLOOP méthode-->**
Cette syntaxe effectue une boucle tant que la méthode retourne Vrai. La méthode admet un paramètre de type Entier long. Elle est appelée une première fois avec la valeur 0 pour permettre une éventuelle phase d'initialisation, puis elle est appelée successivement avec les valeurs 1, 2, 3... tant qu'elle renvoie Vrai.

Pour des raisons de sécurité, la Méthode base Sur authentification Web peut être appelée, une seule fois, avant la phase d'initialisation (exécution de la méthode avec 0 comme paramètre). Si l'authentification est confirmée, la phase d'initialisation a lieu.

En vue de la compilation de la base, il est impératif de déclarer C_BOOLEEN($0) et C_ENTIER LONG($1) au sein de la méthode.

L'exemple de code HTML suivant :

```
<!--#4DLOOP ma_methode-->
<!--#4DVAR var--> <BR>
<!--#4DENDLOOP-->
```

... pourrait se traduire en code 4D par :

```
Si(AuthentificationWebOK)
    Si(ma_methode(0))
        $compteur:=1
        Tant que(ma_methode($compteur))
            ...
            $compteur:=$compteur+1
        Fin tant que
    Fin de si
Fin de si
```

La méthode *ma_methode* pourrait avoir la forme suivante :

```
C_ENTIER LONG($1)
C_BOOLEEN($0)
Si ($1=0)
        `Initialisation
    $0:=Vrai
Sinon
    Si($1<50)
        ...
        var:= ...
        $0:=Vrai
    Sinon
        $0:=Faux   `Arrêt de la boucle
    Fin de si
Fin de si
```

En cas d'erreur d'évaluation, le texte "<!--#4DLOOP expression--> : descriptif" est inséré à la place du contenu situé entre <!--#4DLOOP --> et <!--#4DENDLOOP-->.
Le descriptif de l'erreur peut être l'un des suivants :
- Une expression de ce type n'était pas attendue (erreur générique).

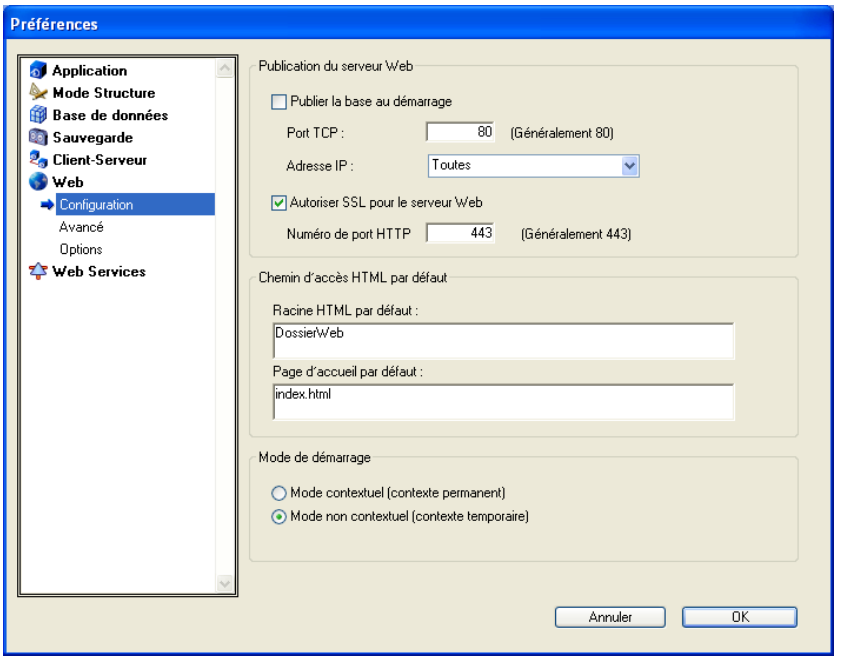
- Nom de table invalide (erreur sur le nom de la table).
- Un tableau était attendu (la variable n'est pas un tableau ou est un tableau à deux dimensions).
- La méthode n'existe pas.
- Erreur de syntaxe (lors de l'exécution de la méthode).
- Erreur de privilège (pas de droits suffisants pour accéder à la table ou à la méthode).
- 4DENDLOOP attendu (le nombre de <!--#4DENDLOOP--> n'est pas égal à celui de <!--#4DLOOP -->).

Référence

Associer des objets 4D à des objets HTML, URLs et actions de formulaires, Utiliser le mode contextuel.

Vous pouvez paramétrer le fonctionnement du serveur Web 4D à l'aide de paramètres définis dans le thème **Web** des Préférences de la base. Cette section décrit les paramètres des pages **Configuration**, **Avancé** et **Options** de ce thème ainsi que la section “Compatibilité Web” de la page **Application/Compatibilité**.

Page Configuration



Publier la base au démarrage

Indique si le serveur Web doit être démarré dès le lancement de l'application 4D. Cette option est détaillées dans la section Mise en route du serveur Web et gestion des connexions.

Port TCP

Par défaut, 4D publie une base Web sur le port TCP standard du Web, qui est le port 80. Si ce port est déjà utilisé par un autre service Web, vous devez changer le port TCP utilisé par 4D pour votre base. La modification du port TCP permet également de lancer le serveur Web 4D sous Mac OS X sans être l'utilisateur root de la machine (cf. section Mise en route du serveur Web et gestion des connexions).

Pour cela, dans la zone de saisie “Port TCP”, indiquez le numéro de port TCP à utiliser pour cette base (c'est-à-dire un numéro de port TCP non utilisé par un autre service TCP/IP sur la machine).

Note : Si vous passez 0, 4D utilisera le numéro de port TCP par défaut, c'est-à-dire 80.

Au niveau du navigateur Web, vous devez inclure ce numéro de port TCP personnalisé dans la description de l'adresse utilisée pour vous connecter à la base Web. L'adresse doit être suivie du signe “deux-points” et du numéro de port. Si, par exemple, vous utilisez le port TCP numéro 8080, vous devrez spécifier dans le browser “123.45.67.89:8080”.

ATTENTION : Lorsque vous utilisez des numéros de port TCP autres que les numéros par défaut (80 pour le mode standard et 443 pour le mode SSL), prenez garde à ne pas spécifier de numéros de port qui se trouvent être les numéros par défaut d'autres services que vous employez simultanément. Si, par exemple, vous envisagez d'exploiter le protocole FTP sur la machine serveur Web, n'utilisez pas les numéros de ports TCP 20 et 21 qui sont les ports par défaut de ce protocole. Pour connaître les attributions standard des numéros de port TCP, vous pouvez vous reporter à la section Annexe B, Numéros des ports TCP dans la documentation de 4D Internet Commands. Les numéros de port inférieurs à 256 sont réservés pour les services standard et les numéros 256 à 1024 sont réservés pour les services spécifiques issus des plates-formes UNIX. Pour une sécurité maximum, il vous suffit de spécifier un numéro de port situé au-delà de ces intervalles, par exemple dans les 2000 ou 3000.

Adresse IP
Vous pouvez définir l’adresse IP sur laquelle le serveur Web doit recevoir les requêtes HTTP. Par défaut, le serveur répond sur toutes les adresses IP (option **Toutes**).
La liste déroulante “Adresse IP” liste automatiquement toutes les adresses IP présentes sur la machine. Lorsque vous sélectionnez une adresse particulière, le serveur ne répond qu’aux requêtes dirigées sur cette adresse.

Cette fonctionnalité est destinée aux serveurs Web 4D hébergés sur des machines ayant plusieurs adresses TCP/IP. C’est, par exemple, fréquemment le cas chez les fournisseurs d’hébergement Internet (*MultiHoming*). Vous pouvez donc placer plusieurs 4D ou 4D Server sur une même machine et de réserver une adresse IP spécifique à chaque base publiée en Internet/Intranet.
La mise en place de cette fonctionnalité nécessite une configuration adéquate de la machine accueillant les différents serveurs Web :

• Configuration MultiHoming sous Mac OS
Pour mettre en place un système utilisant le MultiHoming sous Mac OS :
1. Vous devez disposer d’Open Transport version 1.3 minimum (le MultiHoming est intégré à partir de cette version d’Open Transport).
2. Ouvrez le tableau de bord TCP/IP.
3. Sélectionnez **manuel** dans le pop up menu **Configuration**.
4. Créez un fichier texte nommé "Secondary IP Addresses" et placez-le dans le sous-dossier “Préférences” de votre dossier Système.

Chaque ligne du fichier "Secondary IP Addresses" doit contenir une adresse IP secondaire ainsi que, si nécessaire, un masque de sous-réseau et une adresse de routeur.
Pour plus d'informations sur cette configuration, reportez-vous à la documentation d'Open Transport, fournie par Apple, Inc.

• Configuration MultiHoming sous Windows
Pour mettre en place un système utilisant le MultiHoming sous Windows :
1. Sélectionnez successivement les commandes suivantes :
• Windows NT : Menu **Démarrer** > **Paramètres** > **Panneau de configuration** > icône **Réseau** > onglet **Protocoles** > **Protocole TCP/IP** > bouton **Propriétés** > bouton **Avancé...**
• Windows 2000 : Menu **Démarrer** > **Paramètres** > **Connexions réseau et accès à distance** > **Connexion au réseau local** > Bouton **Propriétés** > **Protocole Internet (TCP/IP)** > Bouton **Propriétés** > Bouton **Avancé...**
• Windows XP : Menu **Démarrer** > **Panneau de configuration** > **Connexions réseau et Internet** > **Connexions réseau** > **Connexion au réseau local** (Propriétés) > **Protocole Internet (TCP/IP)** > Bouton **Propriétés** > Bouton **Avancé...**
La boîte de dialogue de configuration "Paramètres TCP/IP avancés" s'affiche.
2. Cliquez sur le bouton **Ajouter...** dans la zone "Adresses IP" et ajoutez les adresses IP supplémentaires.
Vous pouvez définir jusqu'à 5 adresses IP différentes. Pour cette opération, il se peut que vous ayez besoin de l'assistance d'un administrateur réseau.

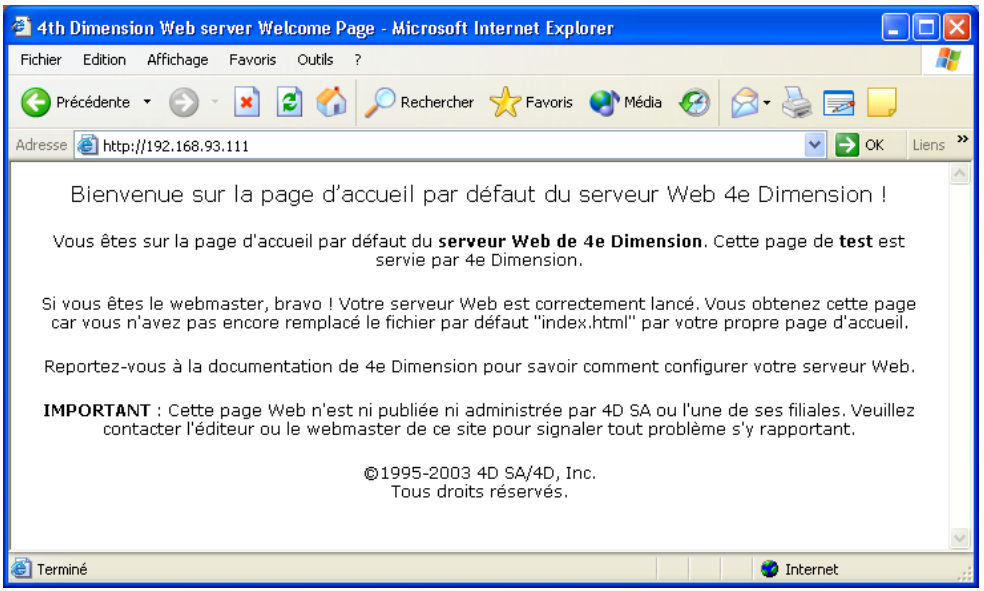
Autoriser SSL pour le serveur Web
Indique si le serveur Web doit ou non accepter les connexions sécurisées. Cette option est décrite dans la section Utiliser le protocole SSL.

Numéro de port HTTPS
Permet de modifier le numéro du port TCP/IP utilisé par le serveur Web pour les connexions HTTP sécurisées via SSL (protocole HTTPS). Par défaut, le numéro du port HTTPS est 443 (valeur standard).
Vous pouvez souhaiter modifier ce numéro pour deux raisons principales :
• par souci de sécurité — en effet, les attaques des pirates contre les serveurs Web se concentrent généralement sur les ports TCP standard, c'est-à-dire 80 et 443.
• sous Mac OS X, pour permettre aux utilisateurs "standard" de lancer le serveur Web en mode sécurisé — en effet, sous Mac OS X, l'utilisation des ports TCP/IP réservés à la publication Web (0 à 1023) requiert des privilèges d'accès spécifiques : seul l'utilisateur "root" peut lancer une application utilisant ces ports. Pour que les utilisateurs autres que "root" puissent lancer le serveur Web, une solution consiste à modifier le numéro du port TCP/IP du serveur Web (cf. section Mise en route du serveur Web et gestion des connexions). Vous pouvez passer toute valeur valide (pour éviter les restrictions d'accès sous Mac OS X, il est nécessaire de passer une valeur supérieure à 1023). Pour plus d'informations sur les numéros de port TCP, reportez-vous ci-dessus au paragraphe "Port TCP".

Racine HTML par défaut
Permet de définir l'emplacement par défaut des fichiers du site Web et indique le niveau hiérarchique sur le disque au-dessus duquel aucune requête ne pourra accéder. Cette option est décrite dans la section Sécurité des connexions.

Page d'accueil par défaut
Cette option permet de désigner la page d'accueil (page "Home") par défaut pour tous les navigateurs se connectant à la base, quel que soit le mode (avec ou sans contexte) défini pour le démarrage des sessions Web. Cette page peut être statique ou semi-dynamique.

Par défaut, lors du premier démarrage du serveur Web, 4D crée une page d'accueil nommée "index.html" et la place dans le dossier racine HTML. Si vous ne modifiez pas ce paramétrage, tout navigateur se connectant au serveur Web obtient la page suivante :



Pour modifier la page d'accueil par défaut, vous pouvez simplement la remplacer par votre propre page "index.html" dans le dossier racine de la base ou saisir dans la zone le chemin d'accès relatif de la page que vous souhaitez définir. Le chemin d'accès doit être établi relativement au dossier racine HTML par défaut. Afin d'assurer la compatibilité multi-plate-forme de vos bases, le serveur Web 4D utilise, pour décrire les chemins d'accès, des conventions d'écriture particulières. Les règles de syntaxe sont les suivantes ;
• les dossiers sont séparés par le caractère /
• le chemin ne doit pas se terminer par /
• pour "remonter" d'un niveau dans la hiérarchie des dossiers, saisissez ".." (point point) devant le nom du dossier
• le chemin ne doit pas commencer par /.
Par exemple, si vous souhaitez que la page d'accueil par défaut soit la page "MyHome.htm", placée dans le dossier "Web" (lui-même situé dans le dossier racine HTML par défaut de la base), saisissez Web/MyHome.htm

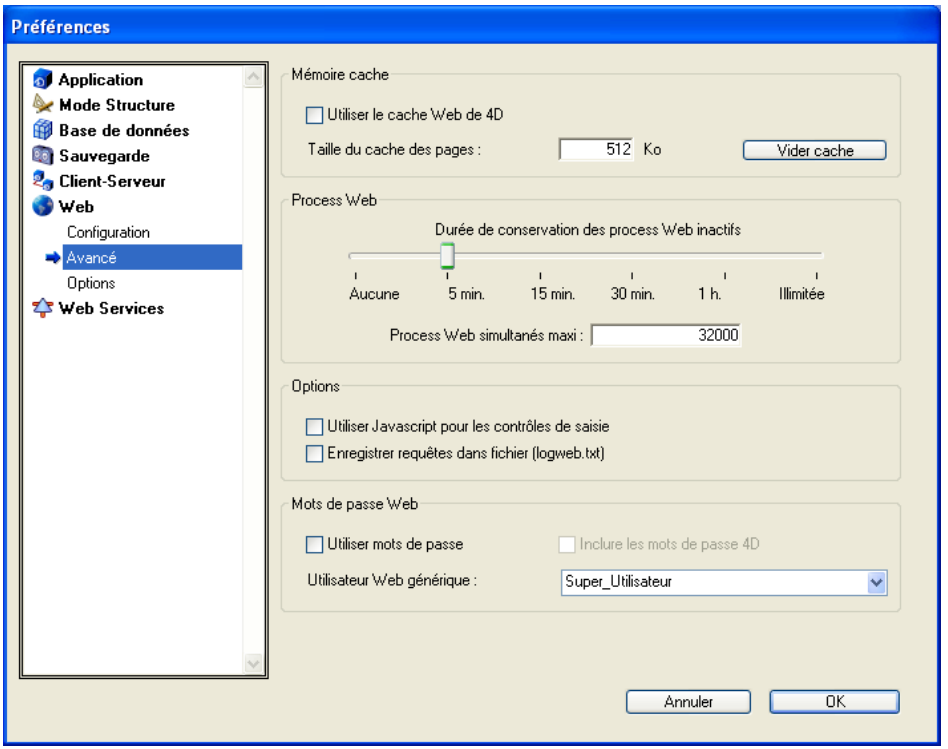
Note : Vous pouvez également définir une page d'accueil par défaut pour chaque process Web à l'aide de la routine FIXER PAGE ACCUEIL.

Si vous ne spécifiez pas de page d'accueil par défaut, le fonctionnement du serveur Web diffère suivant le mode de démarrage :
• Si le serveur Web démarre en mode sans contexte (mode standard), la Méthode base Sur connexion Web est appelée. Il vous revient alors de traiter la requête par programmation.
• Si le serveur Web démarre en mode contextuel, la barre de menus courante — par défaut, la barre de menus n° 1 — est envoyée en tant que page d'accueil.

Mode de démarrage
Permet de définir le mode dans lequel le serveur Web doit démarrer. Cette option est décrite dans la section Utiliser le mode contextuel.

Réutilisation des contextes temporaires (visible uniquement avec 4D Client)
Permet d'optimiser le fonctionnement du serveur Web de 4D Client en recyclant les process Web créés pour le traitement de requêtes Web précédentes.
En effet, le serveur Web de 4D Client a besoin d'un process Web spécifique pour le traitement de chaque requête Web ; lorsque cela s'avère nécessaire, ce process se connecte au poste 4D Server afin d'accéder aux données et au moteur de base de données. Il génère alors un contexte temporaire utilisant ses propres variables, sélections, etc. Une fois la requête traitée, le process est tué. Lorsque l'option **Réutilisation des contextes temporaires** est cochée, 4D maintient les process Web spécifiques créés sur 4D Client et les réutilise pour les requêtes suivantes. L'étape de création du process étant supprimée, les performances du serveur Web sont alors améliorées.
En contrepartie, vous devez veiller dans ce cas à initialiser systématiquement les variables utilisées dans les méthodes 4D afin de ne pas risquer d'obtenir des résultats faussés. De même, il est nécessaire d'effacer les sélections ou enregistrements courants éventuellement définis au cours de la requête précédente.

Page Avancé



Mémoire cache

Le serveur Web 4D dispose d'un cache permettant de charger en mémoire les pages statiques, les images GIF, les images JPEG (<100 ko) et les feuilles de styles (fichiers .css), au fur et à mesure qu'elles sont demandées.
L'utilisation d'un cache permet d'augmenter de manière significative les performances du serveur Web en ce qui concerne l'envoi de pages statiques. Le cache est commun à tous les process Web.
Par défaut, le cache des pages statiques n'est pas activé. Pour l'activer, il suffit de cocher l'option **Utiliser le cache Web de 4D**.
Vous pouvez éventuellement modifier la taille du cache dans la zone **Taille du cache des pages**. La valeur à fixer dépend du nombre et de la taille des pages statiques de votre site Web, ainsi que des ressources dont dispose la machine hôte.

Note : Au cours de l'utilisation de votre base Web, vous pourrez contrôler les performances du cache à l'aide de la routine STATISTIQUES DU CACHE WEB. Si par exemple vous constatez que le taux d'utilisation du cache est proche de 100%, vous pouvez envisager d'augmenter la taille qui lui est allouée.
Les URL particuliers /4DSTATS et /4DHTMLSTATS vous permettent également d'obtenir des informations sur l'état du cache. Reportez-vous à la section Informations sur le site Web.

Une fois le cache activé, lorsqu'une page est demandée par un navigateur, le serveur Web 4D la cherche d'abord dans le cache. Si elle s'y trouve, elle est immédiatement envoyée, sinon le programme charge la page depuis le disque et la place dans le cache. Lorsque le cache est plein et que de la place supplémentaire est requise, 4D "décharge" les pages les moins utilisées, par ordre d'ancienneté.

• Vider le cache
Vous pouvez à tout moment vider le cache des pages et des images qu'il contient (par exemple si vous avez effectué des modifications sur une page statique et souhaitez qu'elle soit rechargée dans le cache). Pour cela, il vous suffit de cliquer sur le bouton **Vider cache**. Le cache est alors immédiatement vidé.

Durée de conservation des process Web inactifs
Permet de définir le délai avant fermeture (*timeout*) des process de connexion Web (mode contextuel uniquement). Cette option est décrite dans la section Utiliser le mode contextuel.

Process Web simultanés maxi
Cette option indique la limite strictement supérieure du nombre de process Web de tout type (contextuels, non contextuels ou appartenant à la "réserve" de process) pouvant être simultanément ouverts sur le serveur. Ce paramètre permet de prévenir la saturation du serveur Web 4D pouvant se produire lors d'un envoi massif de requêtes ou d'une demande excessive de création de contextes.
Par défaut, le nombre est de 32 000 (autrement dit, jusqu'à 31 999 process Web peuvent être créés simultanément). Vous pouvez passer toute valeur incluse entre 10 et 32 000.

Lorsque ce nombre maximum (moins un) de process Web concurrents est atteint, 4D ne crée plus de nouveau process et retourne le message "Serveur non disponible" (statut HTTP 503 - Service Unavailable) à toute nouvelle requête.

Note : Le nombre maximum de process Web peut également être défini à l'aide de la commande FIXER PARAMETRE BASE.

• Comment déterminer la valeur à passer ?
En théorie, le nombre maximum de process Web est le résultat de la division Mémoire disponible / Taille de la pile d'un process Web. Une autre solution consiste à visualiser les informations sur les process Web affichées dans l'Explorateur d'exécution : le nombre courant de process Web et le nombre maximum atteint depuis le démarrage du serveur Web sont indiqués.

• A propos de la réserve de process Web
La "réserve" de process Web permet d'augmenter la réactivité du serveur Web en mode sans contexte. Cette réserve est dimensionnée par un minimum (0 par défaut) et un maximum (10 par défaut) de process à recycler. Ces valeurs peuvent être modifiées à l'aide de la commande FIXER PARAMETRE BASE. Lors du changement du nombre maximum de process Web, si celui-ci est inférieur à la limite supérieure de la "réserve", cette limite est alors abaissée au nombre maximum de process Web.

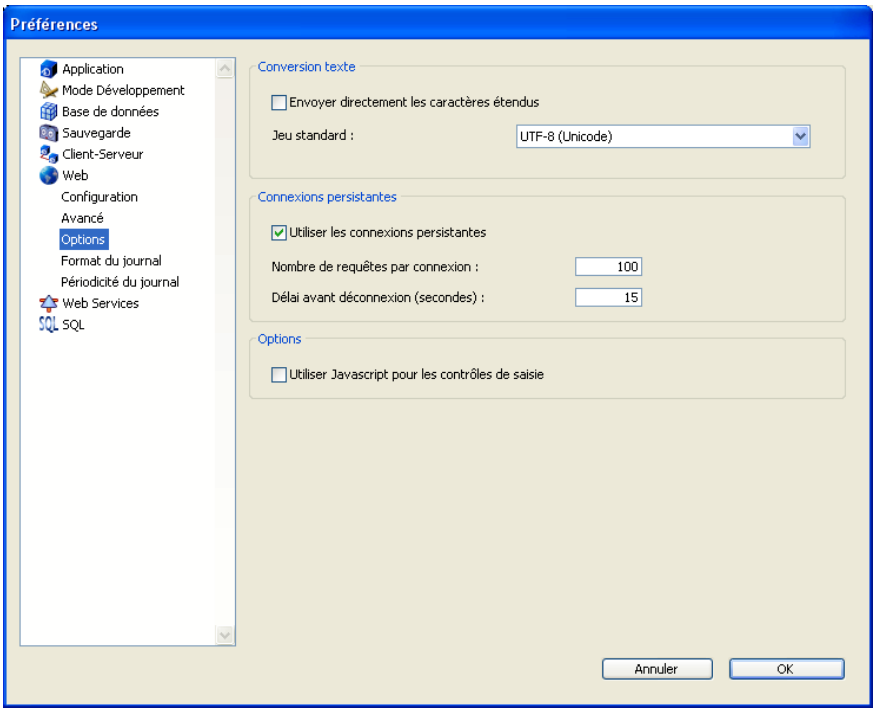
Utiliser Javascript pour les contrôles de saisie
Lorsque cette option est cochée, en mode contextuel une partie des contrôles de saisie peut être effectuée sur les navigateurs via des scripts Java automatiques.
Sur le navigateur, les contrôles de saisie et les types de données (champs ou variables) auxquels ils peuvent s'appliquer sont les suivants :
• valeur minimale (numériques) ;
• valeur maximale (numériques) ;
• valeur obligatoire (numériques et alphas).

Les scripts Java générés, de petite taille, ont pour but d'afficher les boîtes de dialogue d'alerte adéquates sans réellement empêcher la validation (celle-ci reste du ressort de 4D). En effet, si une zone de saisie contient une valeur incorrecte, un message d'alerte est affiché sur le navigateur lorsque l'utilisateur clique sur un bouton (de validation, d'annulation, etc.). Une fois la boîte de dialogue d'alerte validée, si l'utilisateur clique une seconde fois sur le bouton, l'action de celui-ci sera prise en compte. Le contrôle complet de la saisie est effectué par le serveur Web de 4D.

Enregistrer requêtes dans fichier (logweb.txt)
Cette option vous permet de générer sous forme de fichier texte au format CLF un historique des requêtes adressées au serveur Web. Cette option est décrite dans la section Informations sur le site Web.

Zone "Mots de passe"
Paramétrage du système de protection des accès au site Web via des mots de passe. Cette option est décrite dans la section Sécurité des connexions.

Page Options



Envoyer directement les caractères étendus

Par défaut, le serveur Web 4D convertit les caractères étendus présents dans les pages Web (dynamiques et statiques) au normes HTML avant de les envoyer. Ils sont ensuite interprétés par les navigateurs.
Vous pouvez paramétrer le serveur Web de manière à ce que les caractères étendus soient envoyés “tel quels”, sans conversion en entités HTML. Cette option permet un gain de vitesse important sur des systèmes étrangers (principalement japonais).
Pour cela, il suffit de cocher l'option **Envoyer directement les caractères étendus.**

Jeux standard

La liste déroulante **Jeu standard** vous permet de définir le jeu de caractères utilisé par le serveur Web 4D. Par défaut, le jeu de caractères est UTF-8.

Connexions persistantes

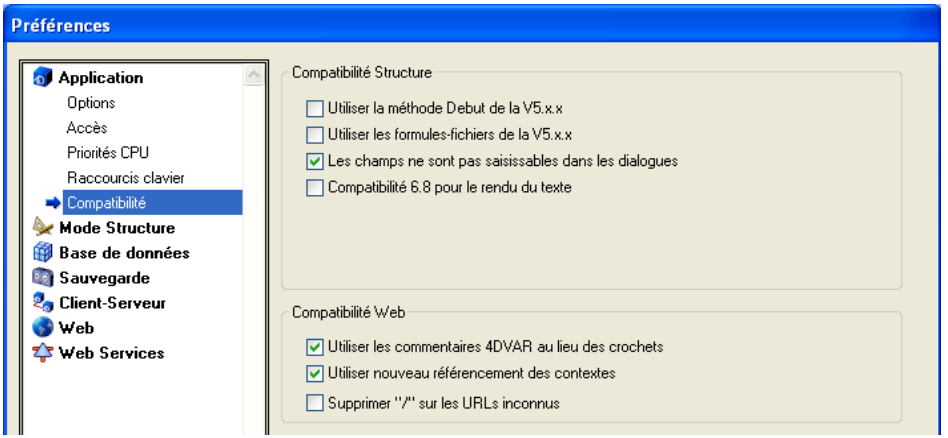
Le serveur Web de 4D peut utiliser des connexions persistantes (*keep-alive*). L’option keep-alive permet de maintenir ouverte une seule connexion TCP pour l’ensemble des échanges effectués par un navigateur et le serveur afin d’économiser les ressources et d’optimiser les échanges.

L'option **Utiliser les connexions persistantes** permet d'activer ou d'inactiver les connexions TCP persistantes pour le serveur Web. Cette option est cochée par défaut. Dans la plupart des cas, il est conseillé de conserver cette option cochée car elle permet d'accélérer les échanges. Si le navigateur Web ne prend pas en charge les connexions keep-alive, le serveur Web 4D bascule automatiquement en HTTP/1.0.
La fonction keep-alive du serveur Web de 4D concerne toutes les connexions TCP/IP (HTTP, HTTPS), en mode contextuel et en mode sans contexte. A noter toutefois que les connexions persistantes ne sont pas systématiquement utilisées pour tous les process Web 4D. Dans certains cas, d'autres mécanismes d'optimisation du serveur Web sont mis en oeuvre. Les connexions persistantes sont principalement efficaces lors de l'envoi de pages statiques.

Deux options permettent de régler le mécanisme des connexions persistantes :
• **Nombre de requêtes par connexion** : permet de définir le nombre maximum de requêtes pouvant transiter dans une même connexion persistante. Limiter le nombre de requêtes par connexion permet d'éviter les risques de saturation du serveur via l'envoi massif de requêtes (technique utilisée par les pirates).
La valeur par défaut (100) peut être réduite ou augmentée en fonction des ressources de la machine hébergeant le serveur Web 4D.
• **Délai avant déconnexion** (timeout) : cette valeur définit le délai maximal (en secondes) pendant lequel le serveur Web maintient ouverte une connexion TCP tant qu'aucune nouvelle requête n'est reçue de la part du navigateur. A l'issue de ce délai, le serveur referme la connexion.
Si le navigateur envoie une requête après la fermeture de la connexion, une nouvelle connexion TCP est automatiquement créée. Ce fonctionnement est transparent pour l'utilisateur.

Compatibilité Web

La page **Compatibilité** du thème **Application** des Préférences comporte des options permettant de régler le fonctionnement du serveur Web :



Utiliser les commentaires 4DVAR au lieu des crochets
Cette option vous permet de définir la notation à utiliser pour l'insertion de variables 4D dans les pages statiques.
• Lorsque l'option est cochée (valeur par défaut), la syntaxe à employer est la notation HTML standard :
<!—#4DVAR MAVAR—> (l'espace entre 4DVAR et le nom de la variable est impératif).
• Lorsque l'option n'est pas cochée, la syntaxe à employer est la notation avec des crochets ([MAVAR]) — solution propriétaire utilisée dans les anciennes versions du serveur Web 4D.

Utiliser nouveau référencement des contextes
Lorsque cette option est cochée (valeur par défaut), en mode contextuel le serveur Web place le numéro de contexte courant dans l'URL de base des documents envoyés aux navigateurs.
Avec le système précédent (option désélectionnée), le serveur Web 4D envoie au navigateur le numéro du contexte pour chaque élément d'une page, ce qui ralentit les traitements. Cette option peut être désélectionnée pour des raisons de compatibilité. Notez qu'après l'avoir modifiée, vous devez redémarrer la base afin que le nouveau fonctionnement soit effectif.

Supprimer "/" sur les URLs inconnus
Dans les versions précédentes de 4D, les URLs inconnus (URLs ne correspondant à aucune page ni URLs spécial) étaient retournés dans les méthodes bases Sur authentification Web et Sur connexion Web ($1) sans débuter par le caractère "/". Cette particularité a été supprimée à compter de 4e Dimension 2004.
Toutefois, si vous avez mis en place des mécanismes basés sur cette particularité et souhaitez conserver le fonctionnement précédent, vous pouvez cocher l'option **Supprimer "/" sur les URLs inconnus**. Par défaut, cette option est cochée pour les bases de données converties et non cochée pour les nouvelles bases.

Référence

FIXER PAGE ACCUEIL, FIXER PARAMETRE BASE, Sécurité des connexions, Utiliser le mode contextuel.

Vous pouvez obtenir diverses informations sur le fonctionnement de votre site Web 4D :

• Vous pouvez contrôler le site par l'intermédiaire d'URL particuliers (/4DSTATS, /4DHTMLSTATS, /4DCACHECLEAR et /4DWEBTEST).
• Vous pouvez générer un historique des requêtes.
• Vous pouvez visualiser la charge du serveur Web dans la page "Evaluation" de l'Explorateur d'exécution de 4D.

URLs de gestion du serveur Web

Le serveur Web 4D accepte quatre URLs particuliers : /4DSTATS, /4DHTMLSTATS, /4DCACHECLEAR et /4DWEBTEST.
• /4DSTATS, /4DHTMLSTATS et /4DCACHECLEAR sont accessibles uniquement au Super_Utilisateur et à l'Administrateur de la base. Si la base ne comporte pas de système de mots de passe 4D, ces URLs sont accessibles à tout utilisateur.
• /4DWEBTEST est toujours accessible.

/4DSTATS

L'URL /4DSTATS renvoie sous forme texte pur :
• le nombre de "hits" (connexions bas niveau),
• le nombre de contextes créés,
• le nombre de contextes n'ayant pas pu être créés,
• le nombre d'erreurs de mots de passe,
• le nombre de pages stockées dans le cache,
• le pourcentage du cache utilisé,
• la liste des pages et des fichiers GIF ou JPEG stockés dans le cache des pages statiques (*).

(*) Pour plus d'informations sur le cache des pages statiques et des images, reportez-vous à la section Paramétrages du serveur Web.

Ces informations peuvent vous permettre de contrôler le fonctionnement de votre serveur et éventuellement d'adapter les paramètres correspondants.

Note : La commande STATISTIQUES DU CACHE WEB vous permet également d'obtenir des informations sur l'utilisation du cache pour les pages statiques.

/4DHTMLSTATS

L'URL /4DHTMLSTATS renvoie, sous forme de texte pur également, les mêmes informations que l'URL /4DSTATS, à la différence près que, dans la dernière rubrique (contenu du cache), seule la liste des pages HTML — donc sans les fichiers .GIF et .JPEG — présentes dans le cache est fournie.

/4DCACHECLEAR

L'URL /4DCACHECLEAR provoque l'effacement immédiat du cache des pages statiques et des images. Il permet donc de "forcer" la mise à jour de pages ayant été modifiées.

/4DWEBTEST

L'URL /4DWEBTEST permet de contrôler le statut du serveur Web. Lorsque cet URL est appelé, 4D retourne un fichier texte contenant les champs HTTP suivants :

• **Date** : date du jour au format RFC 822
Exemple : "Date: Fri, 7 Feb 2003 13:12:50 GMT"
• **Server** : 4D WebStar_D/numéro de version interne
Exemple : "4D WebStar_D/7.0"
• **User-Agent** : nom et version @ adresse IP du client
Exemple : "Mozilla/4.08 (Macintosh; I; PPC, Nav) @ 192.193.00.00"

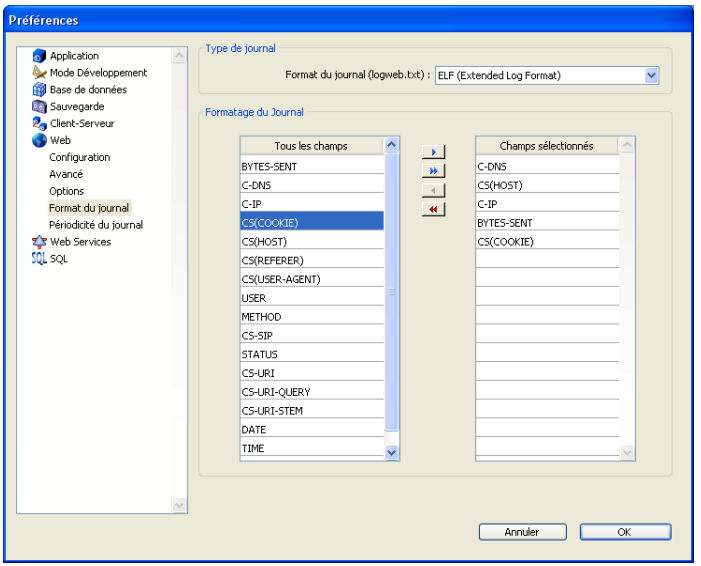
Historique des requêtes

4D vous permet de générer un historique des requêtes.

Ce fichier est nommé "logweb.txt" et est automatiquement placé :
• avec 4D en mode local et 4D Server, dans le dossier **Logs** situé à côté du fichier de structure de la base.
• avec 4D en mode distant, dans le sous-dossier **Logs** du dossier base 4D client (dossier de cache).

Activation et format

L'activation et la configuration du contenu du fichier d'historique s'effectue dans les Préférences de l'application, page Web/Format du journal :



Note : L'activation et la désactivation du fichier d'historique des requêtes peut également être effectuée par programmation, à l'aide de la commande FIXER PARAMETRE BASE.

Le menu de format du journal propose les options suivantes :
• **Pas de journal** : lorsque cette option est sélectionnée, 4D ne génère pas d'historique des requêtes.
• **CLF (Common Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format CLF. Avec le format CLF, chaque ligne du fichier représente une requête sous la forme :
hôte rfc931 utilisateur [JJ/MMM/AAAA:HH:MM:SS] "requête" statut longueur
Chaque champ est séparé par un espace, chaque ligne se termine par la séquence CR/LF (caractère 13, caractère 10).
- hôte : adresse IP du client (ex. 192.100.100.10)
- rfc931 : information non gérée par 4D, c'est toujours - (signe moins)
- utilisateur : nom de l'utilisateur tel qu'il s'est authentifié, sinon - (signe moins). Si le nom de l'utilisateur contient des espaces, ils sont remplacés par des _ (tiret bas).
- JJ : jour, MMM : mois abrégé en 3 lettres et toujours en anglais (Jan, Feb, ...), AAAA : année, HH : heure, MM : minutes, SS : secondes
La date et l'heure sont locales au serveur
- requête : requête envoyée par le client (ex. GET /index.htm HTTP/1.0)
- statut : réponse donnée par le serveur.
- longueur : taille des données renvoyées (hors en-tête HTTP) ou 0.
Les valeurs possibles de statut sont :
200 : OK
204 : Pas de contenu
302 : Redirection
400 : Mauvaise requête
401 : Authentification requise
404 : Non trouvé
500 : Erreur interne
Le format CLF ne peut pas être personnalisé.
• **DLF (Combined Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format DLF. Le format DLF est semblable au format CLF dont il reprend exactement la structure. Il contient simplement deux champs HTTP supplémentaires à la fin de chaque requête : Referer et User-agent.
- Referer : contient l'URL de la page pointant vers le document demandé.
- User-agent : contient le nom et la version du navigateur ou du logiciel client à l'origine de la requête.
Le format DLF ne peut pas être personnalisé.
• **ELF (Extended Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format ELF. Le format ELF est largement répandu dans le monde des serveurs HTTP. Il permet de construire des historiques sophistiqués, répondant à des besoins spécifiques. Pour cette raison, le format ELF est personnalisable : il est possible de choisir les champs à enregistrer ainsi que leur ordre d'insertion dans le fichier.

• **WLF (WebStar Log Format)** : lorsque cette option est sélectionnée, l'historique des requêtes est généré au format WLF. Le format WLF a été développé spécifiquement pour le serveur 4D WebSTAR. Il est semblable au format ELF, il dispose simplement de champs supplémentaires. Comme le format ELF, il est personnalisable.

Configurer les champs
Lorsque vous choisissez le format ELF (Extended Log Format) ou WLF (WebStar Log Format), la zone "Formatage du journal" affiche les champs disponibles pour le format. Vous devez sélectionner chaque champ à inclure dans l'historique. Pour cela, utilisez les flèches de commande ou procédez par glisser-déposer.

Note : Il n'est pas possible de sélectionner deux fois le même champ.

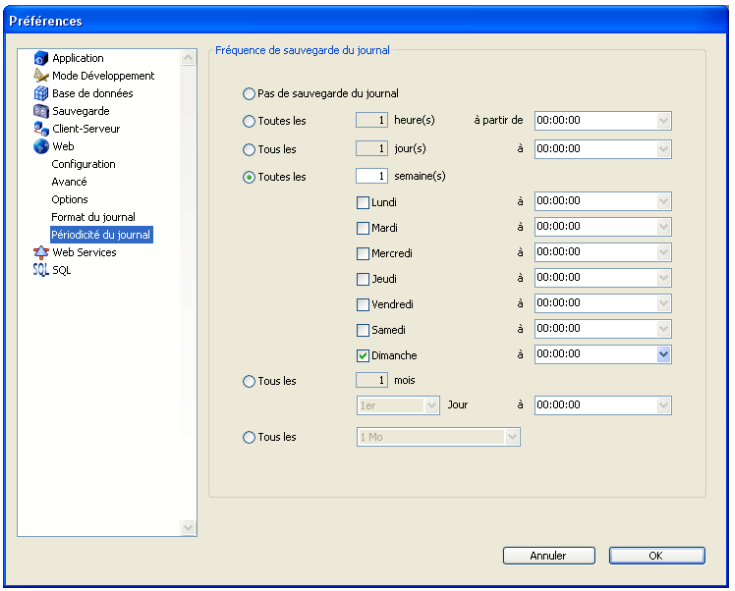
Le tableau suivant liste les champs disponibles pour chaque format (par ordre alphabétique) et décrit son contenu :

Champ	ELF	WLF	Valeur
BYTES_RECEIVED		X	Nombre d'octets reçus par le serveur
BYTES_SENT	X	X	Nombre d'octets envoyés par le serveur au client
C_DNS	X	X	Adresse IP du DNS (ELF : champ identique au champ C_IP)
C_IP	X	X	Adresse IP du client (par exemple 192.100.100.10)
CONNECTION_ID		X	Numéro unique de la connexion
CS(COOKIE)	X	X	Informations sur les cookies contenus dans la requête HTTP
CS(HOST)	X	X	Champ Host de la requête HTTP
CS(REFERER)	X	X	URL de la page pointant vers le document demandé
CS(USER_AGENT)	X	X	Informations sur le logiciel et le système d'exploitation du client
CS_SIP	X	X	Adresse IP du serveur
CS_URI	X	X	URI sur lequel la requête est effectuée
CS_URI_QUERY	X	X	Paramètres d'interrogation de la requête
CS_URI_STEM	X	X	Partie de la requête sans les paramètres d'interrogation
DATE	X	X	DD: jour, MMM: abréviation de 3 lettres pour le mois (Jan, Feb,...), YYYY: année
METHOD	X	X	Méthode HTTP utilisée pour la requête adressée au serveur
PATH_ARGS		X	Paramètres de la CGI : chaîne située après le caractère "$"
STATUS	X	X	Réponse fournie par le serveur
TIME	X	X	HH: heure, MM: minutes, SS: secondes
TRANSFER_TIME	X	X	Délai ayant été nécessaire au serveur pour générer la réponse
USER	X	X	Nom d'utilisateur s'il s'est authentifié, sinon - (signe moins). Si le nom d'utilisateur contient des espaces, ils sont remplacés par des _ (traits de soulignement)
URL		X	URL demandé par le client

Note : La date et l'heure sont indiquées en GMT.

Périodicité de sauvegarde
Le fichier d'historique des requêtes Web pouvant atteindre une taille importante, il est possible de mettre en place un mécanisme d'archivage automatique. Le déclenchement de l'archivage peut être basé sur un délai (exprimé en heures, jours, semaines ou mois) ou une taille de fichier ; à chaque échéance, 4D referme et archive automatiquement le fichier d'historique courant et en crée un nouveau.
Lorsque l'archivage du ficher d'historique se déclenche, le fichier d'historique est archivé dans un dossier nommé "Logweb Archives", créé au même niveau que le fichier logweb.txt (c'est-à-dire à côté du fichier de structure de la base).
Le fichier archivé est renommé sur le modèle "DAAAA_MM_JJ_Thh_mm_ss.txt". Par exemple, pour un fichier archivé le 4 septembre 2006 à 15 heures 50 minutes et 7 secondes : "D2006_09_04_T15_50_07.txt"

Paramétrage des sauvegardes
Les paramètres d'archivage automatique de l'historique des requêtes sont définis dans la page Web/Périodicité du journal des Préférences de l'application :



Vous devez dans un premier temps choisir une échelle de fréquence (jours, semaines...) ou le critère de taille limite en cliquant sur le bouton radio correspondant. Vous devez ensuite éventuellement
préciser le moment de la sauvegarde.
• **Pas de sauvegarde du journal** : la fonction de sauvegarde périodique est inactivée.
• **Toutes les N heure(s)** : cette option permet de programmer la sauvegarde sur une base horaire. Vous pouvez saisir une valeur comprise entre 1 et 24.
- **à partir de** : permet de définir l'heure à laquelle débutera la première sauvegarde horaire.
• **Tous les N jour(s) à N** : cette option permet de programmer la sauvegarde sur une base journalière. Saisissez 1 si vous souhaitez une sauvegarde quotidienne. Lorsque vous cochez cette option, vous devez indiquer l'heure à laquelle la sauvegarde doit être déclenchée.

• **Toutes les N semaine(s), jour à N** : cette option permet de programmer la sauvegarde sur une base hebdomadaire. Saisissez 1 si vous souhaitez une sauvegarde hebdomadaire. Lorsque vous cochez cette option, vous devez indiquer le ou les jour(s) de la semaine et l'heure à laquelle chaque sauvegarde doit être déclenchée. Vous pouvez cocher un ou plusieurs jour(s) de la semaine. Par exemple, vous pouvez utiliser cette option pour définir deux sauvegardes hebdomadaires : une le mercredi et une le vendredi.
• **Tous les N mois, Ne jour à N** : cette option permet de programmer la sauvegarde sur une base mensuelle. Saisissez 1 si vous souhaitez une sauvegarde mensuelle. Lorsque vous cochez cette option, vous devez indiquer le jour de chaque mois auquel la sauvegarde doit être déclenchée,
ainsi que l'heure de déclenchement.
• **Tous les N Mo** : cette option permet de programmer la sauvegarde sur la base de la taille du fichier d'historique des requêtes courant. La sauvegarde est automatiquement déclenchée lorsque le fichier atteint la taille définie. Vous pouvez fixer une taille limite de 1, 10, 100 ou 1000 Mo.

Note : En cas de sauvegarde périodique, si le serveur Web n'était pas lancé au moment théorique de la sauvegarde, 4D considère au lancement suivant que la sauvegarde a échoué et applique les paramétrages adéquats, définis dans les Préférences de la base.

Informations fournies par l'Explorateur d'exécution de 4D

La page Evaluation (rubrique "Informations") de l'Explorateur d'exécution affiche différentes informations concernant le fonctionnement du serveur Web 4D :

• **Occupation du cache Web** : indique le nombre de pages présentes dans le cache Web ainsi que son pourcentage d'utilisation. Cette information n'est disponible que si le serveur Web est actif et si la taille du cache est différente de 0.

• **Temps d'activité du serveur Web** : indique la durée de fonctionnement (au format heures:minutes:secondes) du serveur Web. Cette information n'est disponible que si le serveur Web est actif.

• **Nombre de requêtes http** : indique le nombre total de requêtes HTTP reçues depuis le démarrage du serveur Web, ainsi que le nombre instantané de requêtes par secondes (mesure prise entre deux mises à jour de l'Explorateur d'exécution). Cette information n'est disponible que si le serveur Web est actif.

Référence

Paramétrages du serveur Web, STATISTIQUES DU CACHE WEB.

Le serveur Web de 4D peut fonctionner dans deux modes distincts : le **mode sans contexte** (mode standard) et le **mode contextuel**. Cette section présente ces deux modes et détaille les particularités du mode contextuel.

Attention : Le mode contextuel peut être utilisé avec 4D en mode local et 4D Server uniquement. Le serveur Web de 4D en mode distant ne prend pas en charge ce mode.

Note : La section Premiers pas présente un exemple complet de publication d'une base en mode contextuel.

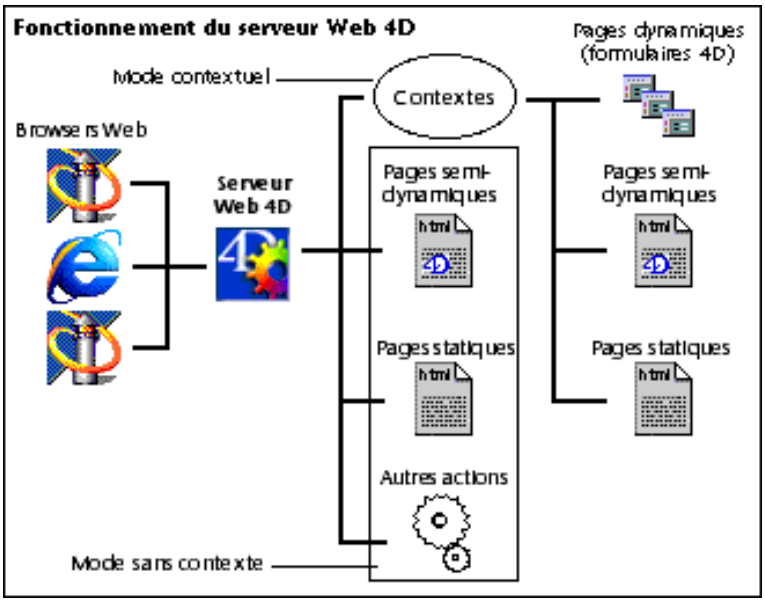
Mode sans contexte et mode contextuel

Le serveur Web de 4D utilise par défaut le **mode sans contexte** (mode déconnecté). Dans ce mode, le fonctionnement du serveur Web 4D est comparable à celui des serveurs Web standard : lorsqu'il reçoit une requête HTTP d'un navigateur (URL, formulaire posté, etc.), le serveur traite la requête puis retourne éventuellement une réponse (envoi d'une page Web par exemple). Aucune connexion spécifique n'est maintenue par la suite entre le serveur et le navigateur.
En mode sans contexte, le serveur Web peut envoyer des pages statiques ou des pages semi-dynamiques. Les pages semi-dynamiques permettent d'accéder aux données de la base ou d'effectuer tout type de traitement via des balises 4D spéciales, évaluées au moment de l'envoi de la page. Les pages semi-dynamiques vous permettent de construire, gérer et envoyer des pages Web dont le contenu est en totalité ou en partie issu d'un traitement effectué par 4D. Le mode sans contexte permet généralement de répondre à la plupart des besoins de développement de sites Web.

En **mode contextuel**, la connexion d'un navigateur Web provoque la création d'un contexte, dans lequel il dispose de sa propre sélection courante, de ses variables, etc. En quelque sorte, chaque navigateur est considéré comme un 4D Client se connectant à la base en mode Application. Le contexte est pris en charge par un process de connexion Web spécifique.
Ce mode permet de publier instantanément une base 4D sur le Web, sans qu'il soit nécessaire de créer des pages Web : 4D génère et envoie au navigateur des pages dynamiques, issues de la conversion automatique en HTML des barres de menus et des formulaires de votre base. En mode contextuel, il reste toutefois possible d'envoyer des pages semi-dynamiques ou des pages statiques. Il est également parfaitement possible d'insérer du code HTML ou Javascript dans les formulaires 4D afin d'ajouter des fonctions aux pages affichées sur le Web. En outre, dans ce mode 4D gère automatiquement les accès simultanés aux données : lorsqu'un navigateur ou un poste 4D Client charge un enregistrement, 4D le verrouille de manière

transparente pour les autres utilisateurs, qu'ils soient navigateurs ou postes 4D Client. De plus, 4D vous permet d'effectuer la saisie de données au sein d'une transaction avec un navigateur Web, comme avec 4D. Ce système permet au serveur Web 4D de contrôler parfaitement les actions des navigateurs et de garantir l'intégrité des données.
En contrepartie de cette facilité de publication, le mode contextuel inclut quelques contraintes :
- les navigateurs Web permettent de "surfer" d'une page Web à une autre, d'un site à un autre, etc. Avec une base de données en client/serveur, cette navigation doit être contrôlée afin de respecter la logique des transactions de la base. Chaque saisie effectuée par un utilisateur dans un enregistrement doit être validée ou annulée afin de ne pas rester dans un état incertain. Le moteur de serveur Web 4D contient des mécanismes automatiques de gestion des sessions et des contextes de la base. Ces mécanismes empêchent l'utilisation de certaines fonctions standard des navigateurs (**Recharger**, **Page précédente**, etc., cf. paragraphe suivant).
- le process de connexion chargé de maintenir le contexte reste actif jusqu'à ce que la période d'inactivité (timeout) du navigateur spécifiée dans les Préférences de la base soit atteinte. Si par exemple le navigateur a quitté le site entre-temps, le contexte est alors "gaspillé".
Ces contraintes destinent plutôt le mode contextuel à une utilisation en Intranet ou dans le cadre d'applications Internet spécifiques.

Le principe de fonctionnement du serveur Web 4D est résumé dans le schéma suivant :

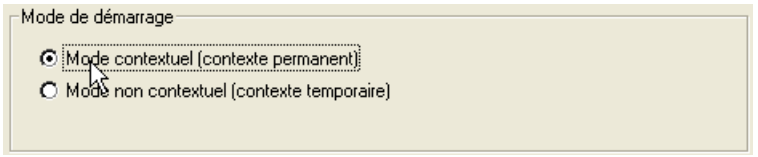


Choix du mode

Le choix du mode de fonctionnement du serveur Web de 4D s'effectue soit :
• au démarrage du serveur, via l'option **Mode de démarrage** des Préférences,
• au cours de l'utilisation du serveur Web, en fonction des URLs envoyés ou des commandes exécutées.
En effet, certains URLs spéciaux et certaines commandes 4D provoquent le changement du mode. Le principe est que le mode courant reste utilisé tant qu'aucun URL ou aucune commande 4D ne provoque un changement de mode.

• Définir le mode contextuel au démarrage

Par défaut, le serveur Web démarre en mode sans contexte. Vous pouvez démarrer le serveur Web directement en mode contextuel. Cela signifie que lorsqu'un utilisateur se connecte à la base, un contexte est automatiquement généré.
Pour définir le mode sans contexte au démarrage, cochez l'option **Mode contextuel (contexte permanent)** dans le thème **Web**, page **Configuration** de la fenêtre des Préférences de la base :



• Commandes et URLs provoquant le changement de mode

Au cours de l'exploitation de la base, vous pouvez changer de mode lors de l'appel des éléments suivants :
• Passage en mode sans contexte :
- ENVOYER BLOB HTML en passant Vrai dans le paramètre optionnel sansContexte
- ENVOYER TEXTE HTML en passant Vrai dans le paramètre optionnel sansContexte
- ENVOYER REDIRECTION HTTP
- URL débutant par /4DACTION
• Passage en mode contextuel :
- URL débutant par /4DMETHOD/MaMéthode.
Lorsque l'URL /4DMETHOD/MaMéthode est envoyé, le serveur Web 4D crée un nouveau contexte et effectue les opérations suivantes :
- la Méthode base Sur authentification Web est exécutée (si elle existe),
- la Méthode base Sur connexion Web est exécutée (si elle existe) — dans ce cas particulier, $1 est égal à /4DMETHOD/MaMéthode au lieu de / (barre oblique),
- enfin, la méthode demandée est exécutée dans le contexte nouvellement créé.

Connaître le nombre de contextes générés

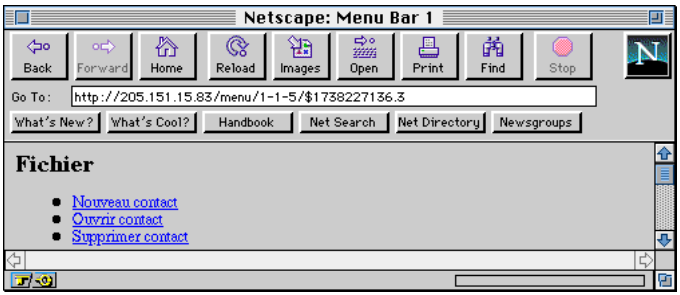
En fonction des actions qu'ils effectuent, certains process Web utilisent des contextes Web, d'autres n'en utilisent pas.

Vous pouvez connaître le nombre de contextes générés, à l'aide de la commande INFORMATIONS PROCESS : pour tout process Web, cette commande indique dans le paramètre origine s'il utilise un contexte (-11, Process web avec contexte) ou non (-3, Process web sans contexte).

Gestion des contextes et de la navigation

Numéro de contexte

Le numéro du process de la connexion Web s'appelle le **numéro de contexte**. C'est un nombre, généré aléatoirement, qui identifie chaque connexion Web. Le numéro de contexte est géré par 4D et par le navigateur pendant toute la connexion Web. Dans l'exemple ci-dessous, le numéro de contexte est 1738227136. Dans la fenêtre de votre navigateur Web, le numéro de contexte apparaît dans l'URL affiché dans la zone de destination :



Les URLs sont automatiquement gérés par 4D pendant toute la session Web en mode contextuel. Chaque fois qu'une requête HTTP est reçue via TCP/IP, 4D extrait le numéro de contexte de l'URL, et peut donc réacheminer la requête au bon process de connexion Web.

Les numéros de contexte permettent à 4D :

- de gérer les sessions Web et celles de la base de données,
- de gérer de façon transparente plusieurs connexions Web simultanées,
- d'éviter des connexions indésirables futures utilisant des marqueurs "Favoris", car un numéro de contexte différent est généré à chaque connexion.

Synchronisation des sessions Web et de la base : numéro de sous-contexte de la connexion Web

Dans la fenêtre précédente, vous pouvez constater que le numéro de contexte est suivi d'un point et d'un autre chiffre. Ce second chiffre s'appelle le **numéro de sous-contexte**. 4D gère et incrémente automatiquement ce numéro chaque fois qu'une nouvelle page HTML issue de 4D est envoyée au navigateur en mode contextuel. Le numéro de sous-contexte est essentiel à la gestion de la session de la base.

Généralement, un navigateur Web dispose de contrôles de navigation comme des boutons du type 'Page précédente' ou 'Page suivante', des fenêtres d'historique, etc. Ces contrôles sont utiles lorsque vous consultez des documents, des notes d'information, des listes, etc. Ils sont moins efficaces lorsque vous effectuez une transaction à partir de la base.
Par exemple, si un utilisateur Web ajoute un enregistrement à une table, il faut savoir si la saisie des données a été validée, c'est-à-dire si l'utilisateur Web a cliqué sur le bouton **Annuler** ou **Valider** dans le formulaire 4D. Si, à ce stade, l'utilisateur Web appelle d'autres pages HTML, la saisie reste dans un état incertain. Pour cela, 4D utilise le numéro de sous-contexte afin de synchroniser la session Web côté 4D.
Lorsqu'un formulaire est soumis au navigateur ou qu'une requête HTTP est envoyée à 4D par le navigateur, si une désynchronisation des sessions du Web et de la base est détectée, 4D envoie le message “En utilisant les contrôles de navigation du navigateur, vous n'avez pas validé les données d'un formulaire. 4D va revenir sur ce formulaire afin que vous le validiez ou l'annuliez”. 4D retourne alors à la page Web de saisie à l'aide du numéro de sous-contexte.
La synchronisation est également essentielle pour le process de connexion Web. L'exécution d'une commande telle que, par exemple, AJOUTER ENREGISTREMENT ([...]) doit s'achever correctement pour que l'on puisse continuer à utiliser la base.

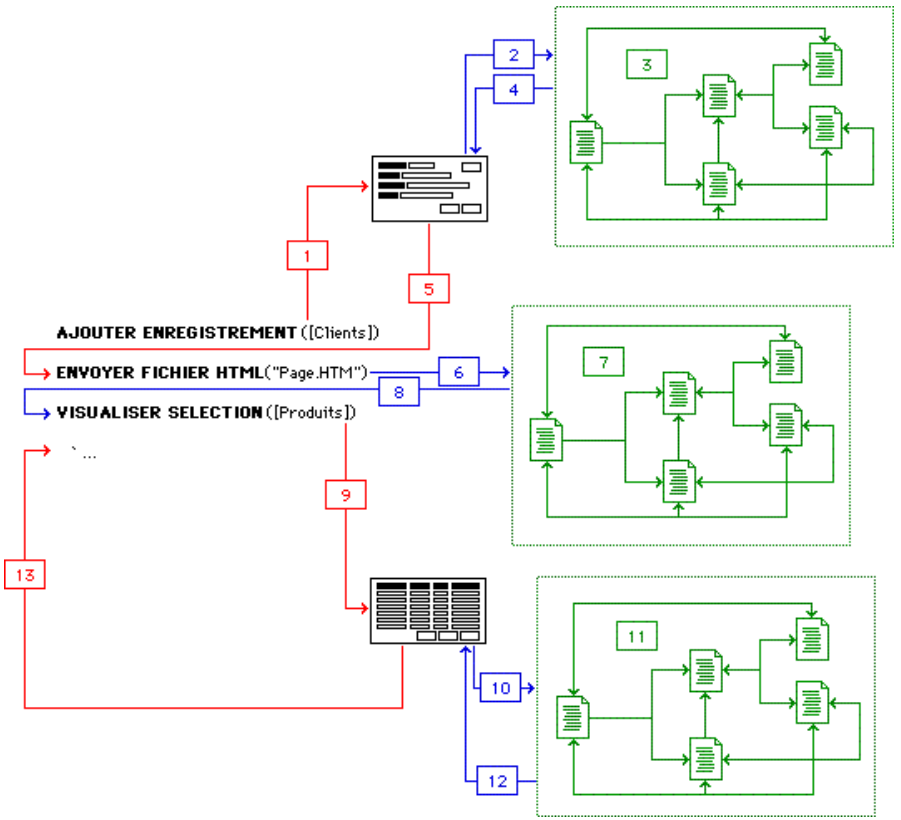
La synchronisation est sélective. Si la page Web courante affichée dans le navigateur est un formulaire 4D (AJOUTER ENREGISTREMENT, VISUALISER SELECTION, DIALOGUE, etc.), les contrôles de synchronisation se manifesteront le cas échéant.
Si la page Web courante est une page HTML statique issue d'un lien présent dans une autre page Web (et envoyée à l'aide de la commande ENVOYER FICHIER HTML), vous pouvez naviguer librement parmi ces pages.

Considérons les lignes de code 4D suivantes :

```
AJOUTER ENREGISTREMENT ([Clients])
ENVOYER FICHIER HTML ("Page.HTM")
VISUALISER SELECTION ([Produits])
```

Le schéma suivant décrit ce qui se passe sur 4D et sur le navigateur Web lors de l'exécution de ce code :

• les lignes rouges signalent les différentes transmissions du formulaire 4D.
• les lignes bleues signalent le transfert entre les pages HTML basées sur 4D et celles qui ne le sont pas.
• les lignes vertes indiquent des pages HTML qui ne sont pas basées sur 4D.



Description des étapes

(1) La commande AJOUTER ENREGISTREMENT est appelée. 4D traduit le formulaire entrée courant de la table en une page HTML et l'envoie au navigateur Web. Si le formulaire contient plusieurs pages, les boutons de navigation standard de 4D permettent de naviguer parmi les pages du formulaire. La navigation est implémentée et effectuée par 4D de façon transparente (via la soumission de formulaire Web).

(2) Pendant la saisie des données (donc dans l'appel à AJOUTER ENREGISTREMENT), l'utilisateur Web clique sur un bouton. La méthode objet du bouton appelle la commande ENVOYER FICHIER HTML.

(3) A la suite de l'appel à ENVOYER FICHIER HTML, si la page HTML contient des liens, l'utilisateur peut naviguer parmi différentes pages. Finalement, lorsque l'instruction ENVOYER FICHIER HTML("") est exécutée, le mode HTML est terminé.

(4) L'exécution de la méthode objet du bouton sur lequel l'utilisateur a cliqué et de la saisie des données liée à AJOUTER ENREGISTREMENT reprennent. Notez que les étapes (2) et (3) peuvent être répétées plusieurs fois pendant la saisie des données.

(5) Enfin, la saisie des données est soit validée soit annulée et le process de connexion Web est exécuté.

(6) L'appel suivant est ENVOYER FICHIER HTML.

(7) Cet étape est semblable à l'étape 3. Si la page HTML contient des liens, il est possible de naviguer parmi plusieurs pages. Lorsque finalement la commande ENVOYER FICHIER HTML("") est appelée, le mode HTML est terminé.

(8) Le process de connexion Web est exécuté.

(9) La commande VISUALISER SELECTION est appelée. 4D traduit le formulaire sortie courant de la table en page HTML et l'envoie au navigateur Web. Pendant le VISUALISER SELECTION, 4D permet de naviguer de façon transparente entre la page de sélection et l'affichage séparé de chaque enregistrement. 4D peut également utiliser MODIFIER SELECTION pour gérer la saisie des données et le verrouillage des enregistrements, par l'intermédiaire des formulaires Web.

(10) Pendant qu'il navigue parmi les enregistrements de la sélection, l'utilisateur clique sur un bouton dans la zone de pied de page du formulaire. La méthode objet du bouton appelle la commande ENVOYER FICHIER HTML.

(11) Cette étape est semblable aux étapes 3 et 7. Si la page HTML contient des liens, il est possible de naviguer parmi plusieurs pages. Lorsque finalement la commande ENVOYER FICHIER HTML("") est appelée, le mode HTML est terminé.

(12) L'exécution de la méthode objet du bouton sur lequel l'utilisateur a cliqué et l'affichage de la sélection liée à AJOUTER ENREGISTREMENT reprennent. Notez que les étapes (10) et (11) peuvent être répétées plusieurs fois pendant la navigation dans la sélection.

(13) Enfin, l'affichage de la sélection est terminé et le process de connexion Web est exécuté.

Et ainsi de suite...

La navigation Web libre (lorsque par exemple vous cliquez sur les boutons Suivant ou Précédent) est possible au sein de tout appel à ENVOYER FICHIER HTML (les zones vertes dans le schéma ci-dessus). En revanche, toute page HTML basée sur 4D (saisie des données, affichage de la sélection..., y compris les boîtes de dialogue standard telles que celles qui sont affichées par les commandes CONFIRMER ou Demander) échappe aux contrôles standard du navigateur. A l'issue de la navigation, 4D va, si nécesssaire, synchroniser les sessions Web avec celles de la base en revenant à la page Web dont le numéro de sous-contexte correspond à celui de la commande en cours d'exécution dans le process de connexion Web.

Process de connexion Web et session Web

De son point de vue, l'utilisateur pilote la session Web à travers ses actions dans le navigateur Web. Du point de vue de la programmation, c'est le process de connexion Web qui pilote la session Web, et non l'inverse. Le navigateur Web affiche les pages envoyées par le process de connexion Web qui :

- soit exécute du code 4D,
- soit attend du navigateur le retour de la page Web courante.

Du point de vue du mode Développement, le process de connexion Web doit être considéré comme un process 4D dont le domaine d'exécution est 4D ou 4D Server, mais dont l'interface utilisateur est située sur le navigateur Web distant. En conséquence, tenez toujours compte de la dualité du process de connexion Web lorsque vous développez des applications pour le Web en mode contextuel. Par exemple :

- Lors de la saisie de tout type de données, la barre de menus principale est celle du navigateur, pas celle de 4D. Dans un formulaire, ne comptez pas sur la barre de menus de 4D ; elle apparaît sur la machine du serveur Web, mais pas sur la machine du navigateur Web.
- Lorsque vous créez des formulaires à destination du navigateur Web, rappelez-vous que les fonctionnalités du formulaire 4D sont restreintes à celles du HTML (avec cependant quelques possibilités supplémentaires liées à 4D). Il n'est donc pas possible d'employer toutes les fonctionnalités des formulaires de 4D (par exemple, tous les types d'objets ou tous les événements formulaires). Pour plus d'informations sur ce point, référez-vous ci-dessous au paragraphe “Conversion HTML automatique”.
- En ce qui concerne la communication interprocess, la commande APPELER PROCESS n'a pas d'effet lorsqu'elle est appliquée à un process de connexion Web, car le formulaire actif est affiché sur le navigateur Web. En revanche, un process de connexion Web peut exécuter un APPELER PROCESS à destination d'un autre process 4D. De plus, la communication interprocess peut s'établir indifféremment dans les deux sens, par l'intermédiaire des commandes LIRE VARIABLE PROCESS et ECRIRE VARIABLE PROCESS, qui ne nécessitent pas la présence d'une interface utilisateur pour le process.

Durée de conservation des process Web inactifs (Timeout**)**

Comme décrit précédemment, un process de connexion Web en mode contextuel soit exécute du code 4D, soit attend le retour de la page Web en cours d'affichage sur le navigateur. Dans ce second cas, le process de connexion Web accordera au navigateur un délai de réponse égal à la valeur de l'option **Durée de conservation des process Web inactifs**, que vous fixez soit dans le thème **Web**, page **Avancé** de la fenêtre des Préférences de la base, soit à l'aide de la commande FIXER TEMPORISATION WEB.
La portée de ce paramétrage est la session de la base. Tous les process de connexion Web contextuels sont assujettis à cette valeur ; ils sont immédiatement affectés lorsque le paramétrage est modifié. La valeur par défaut est 5 minutes.

Note : La commande FIXER TEMPORISATION WEB vous permet de définir une valeur de timeout par process Web.

Vous pouvez augmenter ou réduire cette valeur à votre convenance. Par exemple, vous pouvez augmenter la temporisation si votre application permet aux utilisateurs Web de naviguer vers d'autres sites Web via des liens HTML dans les pages de votre base. En augmentant la temporisation, vous permettez aux utilisateurs de naviguer plus longtemps parmi d'autres sites Web sans fermer les connexions à votre base.

ATTENTION : Il n'est pas possible de stopper par programmation le process de connexion Web. Si vous spécifiez une temporisation longue, le process attendra ce délai même si l'utilisateur Web s'est déconnecté de la base depuis un certain temps. Si vous cochez l'option **Ouverte en permanence**, les process de connexion Web ne s'arrêteront que lorsque vous quitterez la base. A noter toutefois qu'un process de connexion Web est automatiquement tué dès que le serveur Web passe en mode sans contexte.
A la différence du process serveur Web, les process de connexion Web peuvent être tués à l'aide de la commande **Tuer** (disponible dans l'Explorateur d'exécution de 4D lorsque la page Process est affichée).

Conversion HTML automatique

Ce paragraphe précise les éléments, objets et mécanismes pris en charge automatiquement lors de la conversion de la base en HTML par 4D en mode contextuel.

Barres de menus

• Chaque barre de menus est transcrite dans une page HTML. Chaque menu apparaît comme du texte seul et les commandes de menu associées à des méthodes 4D apparaissent comme des liens vers ces méthodes 4D. Les commandes de menu uniquement associées à des actions automatiques apparaissent comme du texte seul.
• Cliquer sur une commande de menu dans le navigateur Web déclenche, dans le process de connexion Web, l'exécution de la méthode 4D associée.

Note : Lorsque la propriété “Démarrer un process” est associée à une commande de menu, la méthode associée est exécutée par le serveur Web de 4D dans un nouveau process de connexion Web, à l'aide de l'URL 4DMETHOD. Dans ce cas, la méthode du menu doit disposer de l'attribut **Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT** (non coché par défaut pour les nouvelles bases). Pour plus d’informations, reportez-vous à la section Sécurité des connexions.
• Une image associée à une barre de menus est placée au-dessous des menus sur le navigateur.

Formulaires

• 4D conserve au maximum la position initiale de chaque objet du formulaire. Notez cependant que le HTML est une application orientée traitement de texte : le positionnement relatif des objets pourra être légèrement différent sur le navigateur.
• Les formulaires multi-pages (y compris la page zéro et les formulaires hérités) sont gérés de manière transparente.
• Les actions automatiques, lorsqu'elles sont cohérentes, sont supportées de manière transparente.
• Les événements formulaires Sur chargement, Sur libération, Sur clic et Sur minuteur sont gérés. Les autres événements ne le sont pas.
• Les balises d'en-tête, de corps, de rupture et de pied de page sont prises en compte lors des appels à VISUALISER SELECTION et MODIFIER SELECTION. L'en-tête du formulaire s'affiche une fois au début de la page HTML, le corps est répété autant de fois que nécessaire et les variables (telles que les boutons) sont placées dans le pied de page à la fin de la page HTML, juste en-dessous des liens automatiques de navigation dans la sélection.
• Les info-bulles des boutons affichés en images dans l’éditeur de formulaires apparaissent sur le navigateur — si celui-ci en permet l’affichage.
• Une image répliquée (affichage **Mosaïque**) insérée en coordonnées (0,0,x,x) dans l’éditeur de formulaires de 4D est envoyée comme image de fond au navigateur. A noter toutefois que les images sombres sont à éviter.

Note : Le serveur Web 4D utilise les CSS1 (Cascading Style Sheet 1) pour générer des pages HTML dont l’apparence est très proche de celle des formulaires obtenus dans 4D. Ces "feuilles de style en cascade", définies par le consortium W3C (World Wide Web Consortium), contiennent un ensemble de caractéristiques définissant l’apparence d’un document : police de caractères, taille, couleur, titres, texte courant, interlignage, etc. Les documents .CSS sont envoyés avec le type MIME “text/css”. En mode contextuel et en mode sans contexte, ces documents ne sont pas analysés par 4D.
Pour des raisons de compatibilité, vous pouvez choisir, à l’aide de la commande FIXER PARAMETRE BASE le mode de conversion Web à utiliser pour les formulaires.

Champs

Lorsqu'un formulaire 4D est converti en page HTML, les champs sont traduits de la manière suivante :

Type champ 4D	Objet HTML	Balise HTML
Alphanumérique	Champ texte (*)	<INPUT Type="text" ...>
Texte	Champ texte (*)	<TEXTAREA ...> (**) <INPUT Type="text" ...> (***)
Réel	Champ texte (*)	<INPUT Type="text" ...>
Entier	Champ texte (*)	<INPUT Type="text" ...>
Entier long	Champ texte (*)	<INPUT Type="text" ...>
Date	Champ texte (*)	<INPUT Type="text" ...>
Heure	Champ texte (*)	<INPUT Type="text" ...>
Booléen	Bouton radio ou case à cocher (*)	<INPUT Type="radio" ...> <INPUT Type="checkbox" ...>
Image	Image (toujours non-saisissable)	<IMG SRC="..." ...>
Sous-table	Pas de support HTML	Aucune
BLOB	Pas de support HTML	Aucune

(*) ou Texte seul si non-saisissable

(**) si la valeur de type Texte se compose de plusieurs lignes

(***) si la valeur de type Texte ne se compose que d'une ligne ou est vide

Note : Les variables saisissables se comportent comme les champs du même type.

Objets de formulaires

Lorsqu'un formulaire 4D est converti en page HTML, les objets de formulaire sont traduits de la manière suivante :

Objet 4D	Objet HTML équivalent	Balise HTML
Ligne	Ligne horizontale (1)	<HR>
Rectangle	Rectangle	Géré par les CSS1
Ovale	Pas de support HTML	Aucune
Rectangle arrondi	Pas de support HTML	Aucune
Image statique	Image ou Image interactive (2)	<IMG SRC="..."> <INPUT Type="image" ...>
Zone de groupe	Texte	Texte avec balises si nécessaire
Texte statique	Texte	Texte avec balises si nécessaire
Bouton	Bouton "submit"	<INPUT Type="submit" ...>
Bouton par défaut	Bouton "submit"	<INPUT Type="submit" ...>
Bouton radio	Bouton radio (3)	<INPUT Type="radio" ...>
Case à cocher	Case à cocher	<INPUT Type="checkbox" ...>

Pop up/Liste déroulante	Liste déroulante	<SELECT ...>...</SELECT>
Combo Box	Liste déroulante	<SELECT ...>...</SELECT>
Zone de défilement	Zone de défilement (4)	<SELECT ...>...</SELECT>
Onglet	Liste d'URLs (5)	<A HREF="/4DTAB/4DVar.Onglet...">
Bouton invisible	Référez-vous à la note 2	
Bouton inversé	Référez-vous à la note 2	
Bouton 3D	Référez-vous à la note 2	
Grille de boutons	Référez-vous à la note 2	
Graphe	Image (non-saisissable)	<IMG SRC="..." ...>
Plug-in	Texte, Image ou Image interactive (6)	Texte avec balises si nécessaire <IMG SRC="..." ...> <INPUT Type="image" ...>

Les objets suivants ne sont pas convertis en HTML et sont donc ignorés :

Liste hiérarchique, Menu déroulant hiérarchique, Sous-formulaire, Bouton radio image, Thermomètre, Règle, Cadran, Menu image, Bouton image, Case à cocher 3D, Bouton radio 3D, Séparateur.

Notes

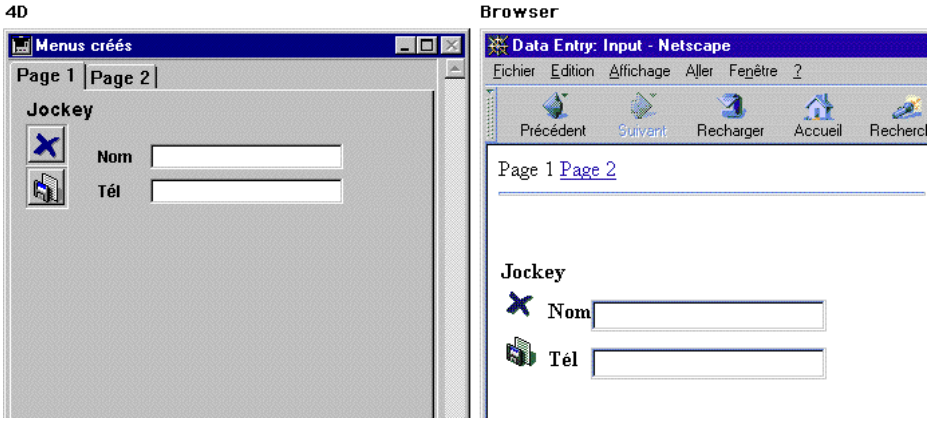
1. Les lignes non horizontales ne sont pas supportées par le HTML ; elles sont donc ignorées.

2. Les boutons non-visibles sont des objets de type Bouton invisible, Bouton inversé, Bouton 3D et Grille de boutons. Lorsqu'une image statique ne chevauche aucun bouton non-visible, elle est traduite comme une image statique. Si elle chevauche au moins un bouton non-visible, elle est traduite comme une "image interactive" gérée sur le serveur (Server-side Image Map).
Sur le navigateur Web, l'image est traitée comme une Server-side Image Map. Du côté de 4D, lorsque le formulaire est reçu, 4D recalcule la position du clic pour générer un événement Sur clic pour le bouton correspondant, comme si l'utilisateur avait réellement cliqué sur le bouton. Gérer des boutons non-visibles est donc très simple, dans la mesure où ils recouvrent ou sont recouverts par des images statiques. Vous contrôlez ces boutons dans la méthode formulaire ou dans leurs méthodes objets, tout comme vous le faites pour l'interface 4D standard. C'est aussi un moyen simple de gérer l'interactivité des images Web (Web Image Mapping). Si un bouton non-visible ne chevauche aucune image statique, il est ignoré lors de la conversion.

3. Le regroupement des boutons radio est maintenu lors de la conversion.

4. Les zones de défilement groupées ne sont pas supportées par le HTML. 4D les traduit en tant que zones de défilement indépendantes situées sur la même ligne.

5. Les onglets (de type tableau ou créés à l'aide de valeurs par défaut définies dans les Propriétés de l'objet) sont convertis sous forme de listes d'URL :



Si les valeurs sont des chaînes vides, 4D affiche 1, 2, 3... sur le navigateur.

6. Les zones de plug-ins sont publiables sur le Web, étant au préalable converties en HTML, en Image ou en "Image Map". Cette dernière solution permet de gérer les clics souris à l'intérieur de la zone de plug-in (par exemple, la zone _AP External clock de 4D_Pack est publiée en Image, le plug-in intégré 4D Chart est publié en Image Map). La manière dont une zone de plug-in incluse dans un formulaire est publiée sur le Web dépend des spécifications de l'éditeur du plug-in.

Visualiser sélection / Modifier sélection

• Le mécanisme UserSet n'est pas supporté.
• Un mécanisme de sélection automatique de pages est fourni par 4D. Pour plus d'informations, reportez-vous à la description de la commande FIXER LIMITES AFFICHAGE WEB.

Commandes 4D

Lorsque vous développez une base 4D pour le Web, vous pouvez vous demander ce qui va se produire lorsque telle ou telle commande sera appelée. La commande produira-t-elle un effet sur la machine du serveur Web ou sur celle du navigateur Web ? Le process de connexion Web s'exécute sur la machine du serveur Web, mais l'interface utilisateur est affichée à distance sur le navigateur Web connecté. Par conséquent, pour le développement d'une base Web, les commandes 4D peuvent être classées de la manière suivante :

Les commandes qui ne sont pas affectées par leur exécution dans un process de connexion Web

Une commande telle que CREER ENREGISTREMENT fonctionne dans le process courant ; dans ce cas, elle crée un enregistrement depuis le process de connexion Web. Le même principe est valable pour les commandes telles que Largeur ecran, qui retourne la largeur de l'écran de la machine du serveur Web (la machine sur laquelle le process est exécuté).

Les commandes qui comportent des fonctionnalités intégrées supplémentaires pour le support transparent du Web

Nom de la commande	**Commentaires**
AJOUTER ENREGISTREMENT	Conversion automatique du formulaire, gestion des formulaires multi-pages
ALERTE	Conversion automatique de la boîte de dialogue
CONFIRMER	Conversion automatique de la boîte de dialogue
DIALOGUE	Conversion automatique du formulaire, gestion des formulaires multi-pages
VISUALISER SELECTION	Conversion automatique du formulaire Mécanisme intégré de pagination de la sélection Le mécanisme UserSet n'est pas géré
MODIFIER ENREGISTREMENT	Conversion automatique du formulaire, gestion des formulaires multi-pages
MODIFIER SELECTION	Conversion automatique du formulaire Mécanisme intégré de pagination de la sélection Le mécanisme UserSet n'est pas géré
CHERCHER	Boîte de dialogue standard de recherche gérée
CHERCHER PAR EXEMPLE	Conversion automatique du formulaire, gestion des formulaires multi-pages
Demander	Conversion automatique de la boîte de dialogue
FIXER MINUTEUR	Support de l'événement formulaire Sur minuteur
REDESSINER	Mise à jour du formulaire affiché sur le navigateur

Les commandes à utiliser avec précaution

Les commandes suivantes s'exécutent localement sur la machine du serveur Web.

Par exemple, vous pouvez demander l'impression d'une sélection à partir du navigateur Web. Cependant, l'impression sera effectuée sur la machine du serveur Web.

De plus, lorsqu'un élément d'interface est appelé, il s'affiche sur la machine du serveur Web, par exemple Ouvrir document(""), par opposition à Ouvrir document("Ce document"). Vous devrez éviter ce genre d'appel car le navigateur Web attendra une réponse jusqu'à ce que la boîte de dialogue soit refermée sur la machine du serveur Web. En revanche, il est tout à fait possible d'appeler ces routines lorsqu'aucune boîte de dialogue n'est requise.

Nom de la commande	Commentaires
Ajouter a document	OK, si aucune boîte de dialogue de fichier n'est appelée
BEEP	Emet un bip sur la machine du serveur Web
Creer document	OK, si aucune boîte de dialogue de fichier n'est appelée
AFFICHER ENREGISTREMENT	Ne fait rien
ECRITURE DIF	OK, si aucune boîte de dialogue de fichier n'est appelée
ECRITURE SYLK	OK, si aucune boîte de dialogue de fichier n'est appelée
EXPORTER TEXTE	OK, si aucune boîte de dialogue de fichier n'est appelée
LECTURE DIF	OK, si aucune boîte de dialogue de fichier n'est appelée
LECTURE SYLK	OK, si aucune boîte de dialogue de fichier n'est appelée
IMPORTER TEXTE	OK, si aucune boîte de dialogue de fichier n'est appelée
CHARGER ENSEMBLE	OK, si aucune boîte de dialogue de fichier n'est appelée
LIRE VARIABLES	OK, si aucune boîte de dialogue de fichier n'est appelée
MESSAGE	Les messages s'affichent sur la machine du serveur Web
Ouvrir document	OK, si aucune boîte de dialogue de fichier n'est appelée
Creer fenetre externe	La fenêtre s'ouvre sur la machine du serveur Web
Ouvrir fichier ressources	OK, si aucune boîte de dialogue de fichier n'est appelée
Creer fenetre	La fenêtre s'ouvre sur la machine du serveur Web
JOUER SON	Le son est joué sur la machine du serveur Web
Imprimer ligne	OK, si aucune boîte de dialogue d'impression n'est appelée
IMPRIMER ETIQUETTES	OK, si aucune boîte de dialogue d'impression n'est appelée
IMPRIMER ENREGISTREMENT	OK, si aucune boîte de dialogue d'impression n'est appelée
IMPRIMER SELECTION	OK, si aucune boîte de dialogue d'impression n'est appelée
QUITTER 4D	Géré, vous pouvez quitter le serveur Web à distance
STOCKER ENSEMBLE	OK, si aucune boîte de dialogue de fichier n'est appelée
ECRIRE VARIABLES	OK, si aucune boîte de dialogue de fichier n'est appelée
FIXER FICHIER HISTORIQUE	OK, si aucune boîte de dialogue de fichier n'est appelée
REGLER SERIE	OK, si aucune boîte de dialogue de fichier n'est appelée (documents)
TRACE	La fenêtre de débogage s'affiche sur la machine du serveur Web

Les commandes qui ne sont pas supportées par les process de connexion Web

Nom de la commande	Commentaires
AJOUTER SEGMENT DE DONNEES	N'appelez PAS cette commande dans un process de connexion Web. Cette commande n'est pas adaptée au Web.
AJOUTER SOUS ENREGISTREMENT	N'appelez PAS cette commande dans un process de connexion Web. Cette commande n'est pas adaptée au Web.
CHANGER UTILISATEUR COURANT	N'appelez PAS cette commande dans un process de connexion Web. Cette commande n'est pas adaptée au Web.
CHANGER PRIVILEGES	N'appelez PAS cette commande dans un process de connexion Web. La fenêtre des mots de passe s'affiche sur la machine de 4D. Le navigateur attendra jusqu'à ce que la fenêtre soit fermée.
GRAPHE SUR SELECTION	N'appelez PAS cette commande dans un process de connexion Web. Cette commande n'est pas adaptée au Web.
MODIFIER SOUS ENREGISTREMENT	N'appelez PAS cette commande dans un process de connexion Web. Cette commande n'est pas adaptée au Web.
TRIER	Gestion sous forme programmée seulement La boîte de dialogue de Tri n'est pas adaptée au Web.
PARAMETRES IMPRESSION	N'appelez PAS cette commande dans un process de connexion Web. Les boîtes de dialogue d'impression s'affichent sur la machine de 4D. Le navigateur attendra jusqu'à ce que les boîtes de dialogue soient fermées.
QR ETAT	N'appelez PAS cette commande dans un process de connexion Web. La fenêtre de l'éditeur d'Etats semi-automatiques s'affiche sur la machine de 4D. Le navigateur attendra jusqu'à ce que la fenêtre soit fermée.

Encapsulation de HTML

Vous pouvez personnaliser le contenu des formulaires 4D convertis en HTML en encapsulant du code HTML (ou du Javascript) dans le formulaire. Le formulaire qui en résulte, côté navigateur Web, est la combinaison d'objets HTML et d'objets 4D.

Insérer une page HTML via un objet texte statique

Un objet texte statique de formulaire 4D contenant par exemple la chaîne "{page.HTM}", insère le document HTML "page.HTM" dans le formulaire 4D à l'endroit où se trouve l'objet texte.

Vous insérez un document dans son intégralité (en fait, tout ce qui est compris entre les balises <BODY> et </BODY>). Vous pouvez soit utiliser un document HTML existant, soit construire par le langage un document que vous sauvegardez sur disque et auquel vous vous référez par la suite.

Note : Dans certains cas, la conversion HTML de formulaires 4D créés en version 6.0.x contenant une référence à un document HTML ({mapage.htm}) peut ne pas donner le résultat escompté avec 4D version 6.7 et suivantes. Dans ce cas, il est possible de modifier le mode de conversion des formulaires à l'aide de la commande FIXER PARAMETRE BASE.

Insérer du code HTML

Toute variable texte 4D peut encapsuler du code HTML dans un formulaire 4D, si son premier caractère a le code 1 (par exemple, vtHTML:=Caractere(1)+"...Code HTML...").
Vous insérez ainsi des morceaux de code. Vous pouvez dans ce cas construire le code HTML en mémoire.

Références de fichiers et URLs

Pour assurer la maintenance du contexte de la base et le numéro d'identification du sous-contexte, 4D recalcule automatiquement les références de fichiers et les URLs.
Par exemple, toutes les références IMG et HREF des fichiers locaux sont recalculées. Si vous insérez votre propre code HTML dans un formulaire 4D avec une variable Texte, vous devez employer la syntaxe décrite plus bas.
Les fichiers locaux GIF sont par exemple recalculés en "/4DBin/_/GIF_file_pathname" où GIF_file_pathname est le nom de chemin complet en HTML vers un fichier GIF, compte tenu de la racine du volume où le fichier est situé.

Une petite méthode 4D comme celle qui suit retourne les références recalculées pour le nom de chemin d'accès reçu en paramètre :

```
      ` Méthode projet WWW Local GIF URL
      ` WWW Local GIF URL ( Texte )
      ` WWW Local GIF URL ( Chemin d'accès natif ) -> URL vers fichier GIF local
C_TEXTE($0;$1)
$0:="/4DBin/_/"+HTML Pathname ($1)
```

Note : Reportez-vous aux exemples de la commande Mac vers ISO pour plus de détails sur la méthode HTML Pathname.

Ensuite, quand vous insérez le code HTML dans un formulaire 4D au moyen d'une variable Texte, vous pouvez écrire :

```
vtHTML:=Caractere(1)+"<P><IMG SRC="+Caractere(34)+WWW Local GIF URL("F:.HTM"
                                +Caractere(34)+"
ALIGN=MIDDLE></P>"+Caractere(13)
```

Cette méthode insèrera le document GIF dans le formulaire 4D à l'endroit où la variable 4D vtHTML est placée.

Important : Vous devez écrire ce type de code uniquement si vous insérez du code HTML dans un formulaire 4D. Si vous envoyez simplement une page HTML avec ENVOYER FICHIER HTML ou ENVOYER BLOB HTML, ou encore si vous utilisez une commande telle que AJOUTER ENREGISTREMENT, souvenez-vous que 4D effectue automatiquement la traduction et le recalcul pour vous.

Le recalcul ne change pas les liens pour les protocoles suivants :
- http:
- ftp:
- mailto:
- news:
- gopher:
- javascript:
- telnet:
- nntp:
- wais:
- prospero:

Référence

FIXER PARAMETRE BASE, Méthode base Sur authentification Web, Méthode base Sur connexion Web, Sécurité des connexions.

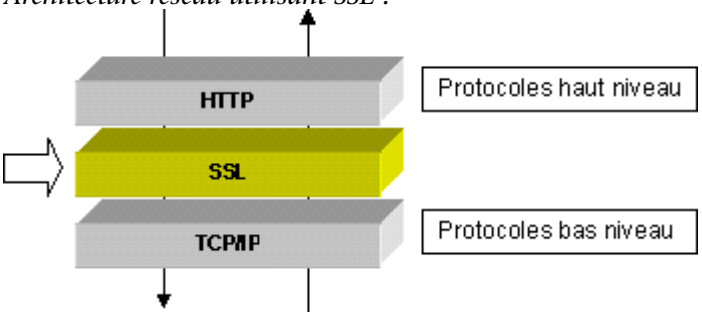
Le serveur Web 4D peut communiquer en mode sécurisé via le protocole SSL (*Secured Socket Layer*).

Qu'est-ce que le protocole SSL ?

Le protocole SSL a pour but de sécuriser les communications entre deux applications — généralement un serveur Web et un navigateur. Ce protocole est largement répandu et compatible avec la plupart des navigateurs Web.

Au niveau de l'architecture réseau, le protocole SSL s'insère entre la couche TCP/IP (bas niveau) et le protocole de haut niveau HTTP, pour lequel il est principalement destiné.

Architecture réseau utilisant SSL :



Note : Le protocole SSL peut également être utilisé pour sécuriser les connexions client/serveur "classiques" de 4D Server ainsi que les connexions du serveur SQL. Pour plus d'informations, reportez-vous à la section Crypter les connexions client/serveur dans le manuel de référence de 4D Server ainsi qu'à la section Configuration du serveur SQL de 4D dans le manuel de référence SQL.

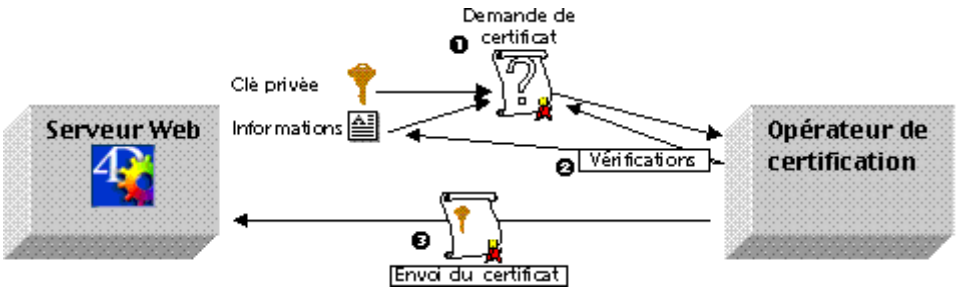
Le protocole SSL permet de garantir l'identité de l'émetteur et du récepteur, ainsi que la confidentialité et l'intégrité des informations échangées :
• **Identification des intervenants** : l'identité de l'émetteur et du récepteur sont confirmées.
• **Confidentialité des informations échangées** : les données envoyées sont cryptées afin de les rendre inintelligibles pour les tiers non autorisés.
• **Intégrité des informations échangées** : les données reçues n'ont pas été altérées, frauduleusement ou accidentellement.

Les principes de sécurisation utilisés par SSL sont basés sur l'emploi d'un algorithme de cryptage utilisant une paire de clés : une clé privée et une clé publique.
La clé privée est utilisée pour crypter les données. Elle est conservée par l'émetteur (le site Web). La clé publique est utilisée pour décrypter les données. Elle est diffusée auprès des récepteurs (les navigateurs Web), via le certificat. L'emploi du SSL dans le cadre d'Internet requiert en effet l'entremise d'un opérateur de certification tel que, par exemple, Verisign®. Moyennant une participation financière du site Web demandeur, cet organisme délivre un certificat, garantissant l'identité du serveur et contenant la clé publique permettant la communication en mode sécurisé.

Note : Pour plus d'informations sur les principes généraux de cryptage et d'emploi de clés publiques/clés privées, reportez-vous à la description de la commande CRYPTER BLOB.

Obtenir un certificat SSL
La mise en place d'un serveur Web 4D fonctionnant en SSL nécessite un certificat numérique délivré par un opérateur de certification. Ce certificat renferme diverses informations dont la carte d'identité du site ainsi que la clé publique utilisée pour communiquer avec lui. Il est transmis aux navigateurs Web se connectant au site. Une fois qu'il est accepté, la communication en mode sécurisé s'établit.

Note : Pour qu'un navigateur accepte les certificats d'une autorité de certification, celle-ci doit être répertoriée dans ses Propriétés.



Le choix de l'autorité de certification dépend de plusieurs facteurs. Plus l'autorité est "connue", plus le nombre de navigateurs acceptant les certificats qu'elle délivre sera important, mais plus le prix à payer sera élevé. Verisign® est une des autorités de certifications les plus utilisées.

Pour obtenir un certificat SSL :

1. Générez une "clé privée" à l'aide de la commande GENERER CLES CRYPTAGE.
Attention : Pour des raisons de sécurité, la clé privée ne doit JAMAIS être diffusée sur un réseau. En fait, elle ne doit pas quitter le poste serveur Web. Le fichier Key.pem doit être placé dans le dossier de la structure de la base.
2. Etablissez une demande de certificat à l'aide de la commande GENERER DEMANDE CERTIFICAT.
3. Envoyez la demande de certificat à l'autorité de certification que vous avez choisie.

Pour remplir la demande de certificat, il vous sera peut-être nécessaire de contacter l'autorité de certification. Les autorités de certification vérifient la réalité des informations qui leur ont été transmises.
La demande de certificat est générée dans un BLOB encrypté au format PKCS. Ce format autorise le copier-coller des clés sous forme de texte et leur envoi par E-mail en toute sécurité, sans risque d'altération de leur contenu. Vous pouvez donc par exemple sauvegarder le BLOB contenant la demande de certificat dans un document texte (à l'aide de BLOB VERS DOCUMENT), puis l'ouvrir et copier-coller son contenu dans un E-mail ou un formulaire Web destiné à l'autorité de certification.
4. Une fois que vous avez reçu votre certificat, créez un fichier texte que vous nommerez "Cert.pem" et copiez dans ce fichier le contenu du certificat.
Vous pouvez recevoir votre certificat sous plusieurs formes (généralement via un E-mail ou un formulaire HTML). Le serveur Web 4D accepte la plupart des formats de texte (Mac OS, PC, Linux...) pour les certificats. En revanche, le certificat doit être au format PKCS.
5. Placez le fichier "cert.pem" dans le dossier contenant la structure de la base.
Le serveur Web peut dès lors fonctionner en mode SSL. La durée de validité d'un certificat varie généralement entre six mois et un an.

Installation et activation du SSL dans 4D

Pour que vous puissiez utiliser le protocole SSL avec le serveur Web de 4D, plusieurs éléments doivent être présents sur le serveur, à différents emplacements :

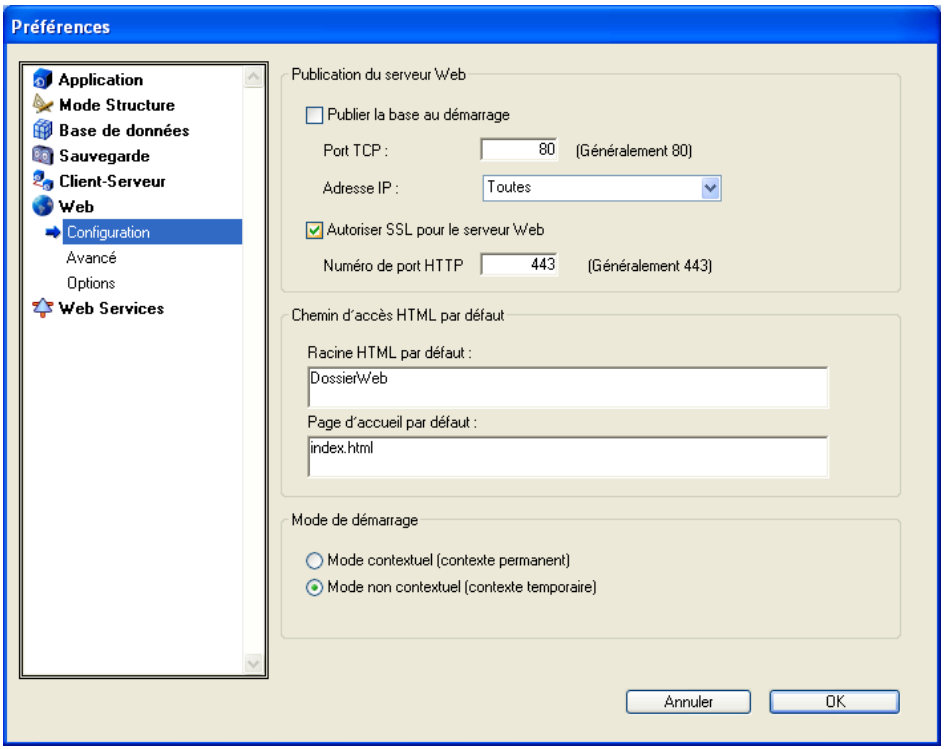
• 4DSLI.DLL : interface de la couche sécurisée (*Secured Layer Interface*) dédiée à la gestion du SSL.
Ce fichier est installé par défaut, il est placé :
- sous Windows, à côté du fichier exécutable de l'application 4D ou 4D Server
- sous Mac OS, dans le sous-dossier [4D Extensions] du progiciel 4D ou 4D Server.

• key.pem : document contenant la clé de cryptage privée.
- avec 4D en mode local ou 4D Server, ce fichier doit se trouver dans le dossier de la base.
- avec 4D en mode distant, ce fichier doit se trouver dans le dossier/progiciel de l'application 4D.

• cert.pem : document contenant le certificat.
- avec 4D en mode local ou 4D Server, ce fichier doit se trouver dans le dossier de la base.
- avec 4D en mode distant, ce fichier doit se trouver dans le dossier/progiciel de l'application 4D.

Note : La présence du composant 4DSLI.DLL est également nécessaire pour l'utilisation des commandes de cryptage des données CRYPTER BLOB et DECRYPTER BLOB.

L'installation de ces éléments rend possible l'utilisation du SSL dans les connexions au serveur Web 4D. Toutefois, pour que les connexions SSL soient acceptées par le serveur Web 4D, vous devez “activer” SSL. Ce paramètre est accessible dans la page **Configuration**, thème **Web** des Préférences :



Par défaut, les connexions SSL sont autorisées. Vous pouvez désélectionner cette option si vous ne souhaitez pas exploiter les fonctionnalités SSL avec votre serveur Web, ou si un autre serveur Web autorisant les connexions sécurisées fonctionne sur le même poste.

Le port TCP utilisé pour les connexions SSL est par défaut le 443. Ce numéro de port peut être modifié dans la zone **Numéro de port HTTPS** afin, par exemple, de renforcer la sécurité du serveur Web (pour plus d'informations sur ce point, reportez-vous à la section Paramétrages du serveur Web). Le port TCP défini dans cette page de Préférences est utilisé pour les connexions du serveur Web en mode standard.

Note : Les autres Préférences de gestion du serveur Web 4D (mots de passe, délai avant déconnexion, taille du cache, etc.) restent appliquées, que le serveur fonctionne en mode SSL ou non.

Connexion des navigateurs en SSL

Pour qu'une connexion Web s'effectue en mode sécurisé, il suffit simplement que l'URL envoyé par le navigateur débute par "https" (au lieu de "http").
Dans ce cas, une boîte de dialogue d'alerte apparaît sur le navigateur. Une fois la boîte de dialogue validée, le certificat est envoyé au navigateur par le serveur Web.

Les deux parties "négocient" alors l'algorithme de cryptage qui va être utilisé pour la connexion.
Le serveur Web 4D dispose de plusieurs algorithmes de cryptage symétriques (RC2, RC4, DES...). L'algorithme commun le plus puissant est utilisé.

Attention : Le niveau de cryptage autorisé est soumis à la législation en vigueur dans le pays d'utilisation.

Contrôle du mode de connexion

L'utilisation de SSL dans le serveur Web 4D ne nécessite pas de configuration système particulière. Toutefois, vous devez tenir compte du fait qu'un serveur Web SSL peut également fonctionner en mode non sécurisé. Le changement de mode de connexion peut s'effectuer si le navigateur en fait la demande (il suffit à l'utilisateur, dans la zone d'URL du navigateur, de remplacer "HTTPS" par "HTTP"). Il revient au développeur d'interdire ou de rediriger les requêtes effectuées en mode non sécurisé — la routine Connexion Web securisee permet de connaître le mode de connexion courant.

Référence

Connexion Web securisee, CRYPTER BLOB, DECRYPTER BLOB, GENERER CLES CRYPTAGE, GENERER DEMANDE CERTIFICAT, Paramétrages du serveur Web.

WML

Le serveur Web 4D supporte la technologie WML (*Wireless Markup Language*). Cette fonctionnalité autorise la consultation et la saisie d'informations dans des bases 4D à l'aide d'un téléphone mobile ou d'un assistant électronique personnel (PDA).

Note : Le langage WML, associé au protocole WAP (*Wireless Application Protocol*), est développé conjointement par plusieurs sociétés. Le WAP propose un ensemble d'outils de communication réseau, destiné à permettre aux téléphones mobiles et aux PDA de visualiser du texte publié dans des pages Web. La technologie WML est ouverte et libre de droits. Pour plus d'informations sur les spécifications du WML, veuillez consulter le site Web de Phone.com : http://www.phone.com/

La visualisation et la saisie d'informations s'effectuent par l'intermédiaire de pages WML utilisant le mécanisme des balises 4DVAR ou 4DSCRIPT.

Voici la liste des documents WML acceptés par le serveur Web 4D

Extension	Type MIME	Description
.wml	text/vnd.wap.wml	Pages WML (toujours analysées par 4D*)
.wmls	text/vnd.wap.wmlscript	Scripts WML (côté client)
.wmlc	application/vnd.wap.wmlc	Pages WML binaires
.wmlsc	application/vnd.wap.wmlscript	Scripts WML binaires
.wbmp	image/vnd.wap.wbmp	Images bitmap destinées aux mobiles (parfois non prises en charge)

* Permet l'insertion dynamique de données via les balises 4DVAR et 4DSCRIPT.

XML

Le serveur Web 4D accepte les documents .xml, .xsl et .dtd. Ces documents sont respectivement envoyés avec le type MIME "text/xml", "text/xsl" et "text/xml".

Quel que soit le mode (contextuel ou sans contexte) depuis lequel ces documents sont envoyés, 4D analyse leur contenu et traite les éventuelles balises de type 4DVAR ou 4DSCRIPT qu'ils contiennent, afin de générer du XML dynamique.

Note : Il n'est pas possible d'envoyer du XML depuis l'intérieur même d'un formulaire 4D en mode contextuel en utilisant une balise du type {mapage.xml} insérée dans un texte statique.

Référence

Associer des objets 4D à des objets HTML.

Le serveur Web 4D permet de tirer parti des CGI (*Common Gateway Interface*). Les CGI sont aux pages Web ce que les plug-ins sont aux méthodes 4D. Appelé par le serveur Web, le CGI exécute une tâche et retourne une réponse — une page Web complète ou du code HTML venant s'insérer dans une page envoyée par le serveur. Des CGI sont fréquemment utilisés pour afficher les compteurs d'accès, gérer les livres d'or (*guestbook*), recevoir le résultat d'un formulaire (*form to mail*), etc. Une multitude de CGI sont aujourd'hui disponibles. La plupart d'entre eux sont dans le domaine public.

Note : Le terme CGI désigne, à l'origine, la norme définissant l'interfaçage des applications externes avec les serveurs HTTP. Par extension, "CGI" est aujourd'hui utilisé pour désigner ces applications externes elles-mêmes.

4D peut utiliser des CGI de deux manières :
• le serveur Web 4D peut exécuter des CGI en mode automatique ou en mode manuel
• le serveur Web 4D peut être interrogé par tout serveur HTTP via des extensions de type CGI et ISAPI (Windows uniquement).

Exécuter des CGI depuis le serveur Web 4D

4D prend en charge tous les types de CGI, sous Mac OS X et sous Windows. Un CGI se présente sous la forme d'un programme exécutable, d'un script PERL ou d'une DLL s'interfaçant avec un serveur Web.
- des **exécutables** (.EXE) utilisant la "console" et les variables d'environnement. Le code source est généralement multi-plate-forme (Windows et Unix). Leurs noms sont de la forme nnnn.exe ou nph-nnnn.exe. Pour plus d'informations sur ce type de CGI, veuillez consulter le site http://hoohoo.ncsa.uiuc.edu/cgi/.
- des **scripts PERL** utilisant la "console" et les variables d'environnement. Ces CGI nécessitent la présence d'un interpréteur permettant de les exécuter. Ils présentent l'avantage d'être entièrement multi-plate-forme (Windows, Unix et Mac OS). Leurs noms sont de la forme nnnn.pl, nph-nnnn.pl, nnnn.cgi ou encore nph-nnnn.cgi.
Pour plus d'informations sur ce type de CGI, veuillez consulter le site http://www.perl.com/.
- des **DLL ISAPI**, c'est-à-dire des extensions pour IIS (*Internet Information Server*). Leurs noms sont de la forme nnnn.dll ou nph-nnnn.dll. Pour des raisons de performances, les DLL appelées de la sorte ne sont déchargées qu'à l'arrêt du serveur Web. Pour plus d'informations sur ce type de CGI, veuillez consulter le site http://www.microsoft.com/iis/.

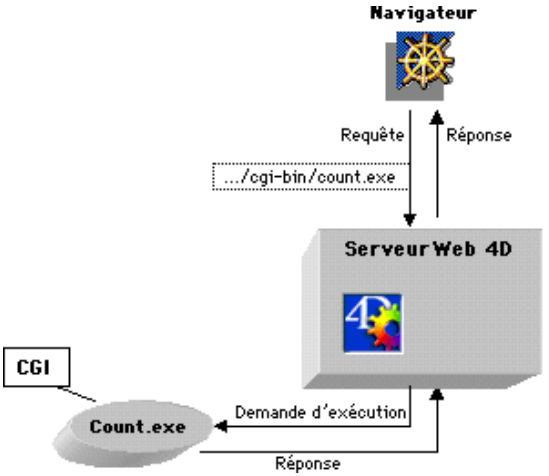
Exécuter des CGIs en mode automatique

L'appel automatique d'un CGI s'effectue par l'intermédiaire d'un URL, d'une action de formulaire ou d'une balise HTML inséré(e) dans une page, en fonction de la tâche effectuée par le CGI. Dans tous les cas, la chaîne HTML doit contenir /cgi-bin/ suivi du nom du CGI et éventuellement d'un chemin d'accès (utilisant la syntaxe HTML) ainsi que d'une chaîne d'interrogation.

Par exemple, l'URL "http://195.1.2.3/cgi-bin/search.exe" provoquera l'exécution du CGI search.exe. De même, si vous placez la balise <IMG SRC="/special/cgi-bin/counter.exe"> au sein d'une page HTML, le CGI counter.exe sera exécuté lors de l'envoi de la page.

Pour pouvoir être appelés, les CGI doivent obligatoirement être placés à la racine d'un dossier nommé **cgi-bin**. Ce dossier doit lui-même être situé à la racine du serveur Web ou dans un sous-dossier. Il peut y avoir plusieurs dossiers **cgi-bin** par serveur. Ce dossier peut contenir d'autres fichiers que des exécutables, mais seuls ces derniers peuvent être appelés depuis un client Web.

Exemple d'installation avec un CGI appelé "Count.exe":



Voici des exemples d'emplacements et les URL pouvant être appelés :

Emplacement des éléments (racine du serveur Web)	URLs correspondants
Dossier [mabase]	
+ mabase.4db (structure)	(http://195.1.2.3/)
Dossier [cgi-bin]	
+ compteur.exe	(http://195.1.2.3/cgi-bin/compteur.exe)
Dossier [Divers]	
+ Dossier [cgi-bin]	
++ script.pl	(http://195.1.2.3/Divers/cgi-bin/script.pl)

Exécuter des CGIs en mode manuel

L'appel de CGIs en mode manuel requiert l'utilisation de la commande FIXER EXECUTABLE CGI. Cette commande permet notamment d'exécuter un CGI sans que celui-ci soit visible pour l'utilisateur Web dans l'URL.
Pour plus d'informations sur ce point, reportez-vous à la description de cette commande.

Interaction entre le serveur Web 4D et les CGI

L'appel d'un CGI ne modifie jamais l'environnement de 4D (sélections, variables...).
4D n'impose aucune limite de taille de la réponse. Cependant, la durée maximum de traitement allouée au CGI est fixée à 30 secondes. Au-delà de ce délai, le serveur Web retournera une erreur.
Un CGI est toujours exécuté hors contexte, quel que soit le mode depuis lequel il a été appelé.
A noter toutefois qu'en mode contextuel, il est conseillé de ne pas utiliser de CGI qui renvoie du code HTML, car il y a dans ce cas risque de désynchronisation du contexte.

Erreurs renvoyées par 4D lors d'un appel CGI

Lorsque l'appel d'un CGI génère une erreur, 4D retourne une des réponses suivantes, sous forme de page HTML standard :
• *Non trouvé* : le CGI n'a pu être localisé par 4D, ou bien l'interpréteur PERL n'est pas installé
• *Interdit* : le client Web demande autre chose qu'un exécutable dans un dossier [cgi-bin]
• *Timeout* : le CGI n'a pu traiter la requête en 30 secondes
• *Mauvaise réponse* : la réponse du CGI n'a pu être traitée par 4D ou la DLL ISAPI a causé une exception
• *Erreur interne* : mémoire saturée, etc.

Note : Lorsqu'un CGI ne fonctionne pas, vérifiez que les privilèges d'exécution du CGI sont suffisants et que les retours à la ligne dans le script CGI sont corrects.

Informations à destination des développeurs de CGI

Ce paragraphe est principalement destiné aux programmeurs souhaitant développer des CGI pour leur bases 4D.

• Variables d'environnement
4D définit les variables d'environnement en conformité avec les spécifications CGI/1.1, avec les précisions suivantes :
GATEWAY_INTERFACE : toujours "CGI/1.1"
SERVER_SOFTWARE : toujours de la forme "4D WebStar_D/version"
SERVER_PROTOCOL : toujours "HTTP/1.0"
SERVER_PORT_SECURE : contient "1" si la connexion HTTP est sécurisée, sinon "0".
PATH_TRANSLATED : contient le chemin d'accès complet de la racine HTML du serveur, auquel est ajouté la portion de chemin qui suit le nom du CGI. Par sécurité, la portion de chemin ne peut contenir les séquences // ou ..
Exemple : Racine du serveur C:/web
Pour un appel CGI du type /cgi-bin/cgi.exe/path, PATH_TRANSLATED vaut "C:/web/path".
Pour un appel CGI du type /cgi-bin/cgi.exe/../path, 4D renvoie l'erreur *Interdit*.

REMOTE_IDENT : nom d'utilisateur, sinon non définie
HTTP_AUTHORIZATION, HTTP_CONTENT_LENGTH et HTTP_CONTENT_TYPE : non définies
ALL_HTTP et URL sont définies dans le cas d'appels de DLL ISAPI.
CERT_xxx et HTTPS_xxx sont définies si la connexion est sécurisée (pour les DLL uniquement).

Note : La commande FIXER VARIABLE ENVIRONNEMENT permet de définir ces variables.

En plus des variables d'environnement standard, 4D ajoute des variables texte du type FORMVAR_nomvariable :
- si la requête est envoyée avec la méthode "POST", ces variables correspondent aux zones de saisie du formulaire (par exemple FORMVAR_NOM, FORMVAR_PRENOM...) à l'exception des champs binaires (INPUT TYPE="FILE"). Ce système peut être utilisé avec les formulaires encodés "www/url-encoded" et "multipart/form-data".
- si la requête est envoyée avec la méthode "GET", ces variables correspondent aux valeurs passées par la chaîne d'interrogation (par exemple, dans le cas de l'URL .../cgi.exe?nom=martin&code=75, FORMVAR_NOM vaudra "martin" et FORMVAR_CODE vaudra "75").

Ce fonctionnement présente l'avantage de faciliter le traitement des formulaires (il n'est pas nécessaire d'analyser les chaînes a=1&b=2&...), mais rend le CGI spécifique à 4D.

• Traitement des réponses retournées par les CGI
Si le nom du CGI (exécutable Windows ou script PERL) débute par nph- (*No Parsing Header*), 4D retourne la réponse "telle quelle" au client Web. Dans ce cas, il revient au CGI de respecter la norme HTTP. En ce qui concerne les DLL ISAPI, 4D n'analyse jamais la réponse, que le préfixe nph- soit présent ou non.

Si ce n'est pas le cas, 4D se charge de renvoyer l'en-tête HTTP :
- si "Content-Type" n'est pas spécifié par le CGI, 4D renvoie systématiquement "Content-Type: text/html",
- si "Location" est spécifié, 4D ignore les autres éléments de la réponse et effectue une redirection HTTP,
- si "Status" n'est pas spécifié, 4D renvoie "HTTP/1.0 200 OK".
4D accepte tout type de changement de ligne (Windows-CRLF, Mac OS-CR, Unix-LF) dans l'en-tête de la réponse HTTP et se charge de la reformater.

Dans le cas des DLL ISAPI, 4D accepte les traitements asynchrones (HttpExtensionProc retourne HSE_STATUS_PENDING). L'appel à ServerSupportFunction (HSE_REQ_DONE_WITH_SESSION) doit avoir lieu dans les 30 secondes. Si la fonction TerminateExtension est définie, elle est toujours appelée avec la valeur HSE_TERM_MUST_UNLOAD.

Interroger le serveur Web 4D via des CGI (Windows uniquement)

4D est accompagné de deux extensions, 4DISAPI.DLL et NPH-CGI4D.EXE. Le but de ces extensions est de permettre à un serveur HTTP de transmettre des requêtes au serveur Web 4D. Avec ces extensions, 4D peut être interrogé par tout autre serveur HTTP. Ce mécanisme autorise par exemple le développement de systèmes dans lesquels un serveur Web 4D non sécurisé peut être interrogé via un autre serveur HTTP, tournant lui en mode sécurisé.

Note : Ces deux extensions sont disponibles sous Windows uniquement.

• L'extension 4DISAPI.DLL respecte les spécifications définies par ISAPI (*Internet Services Application Programming Interface*). La technologie ISAPI a été développée à l'origine par Microsoft® pour le serveur IIS, mais a été depuis rendue compatible avec de nombreux serveurs HTTP tels que Netscape®, Apache® ou Sambar®.

• L'extension NPH-CGI4D.EXE respecte les spécifications des CGI (*Common Gateway Interface*) et peut être utilisée avec l'ensemble des serveurs compatibles CGI.

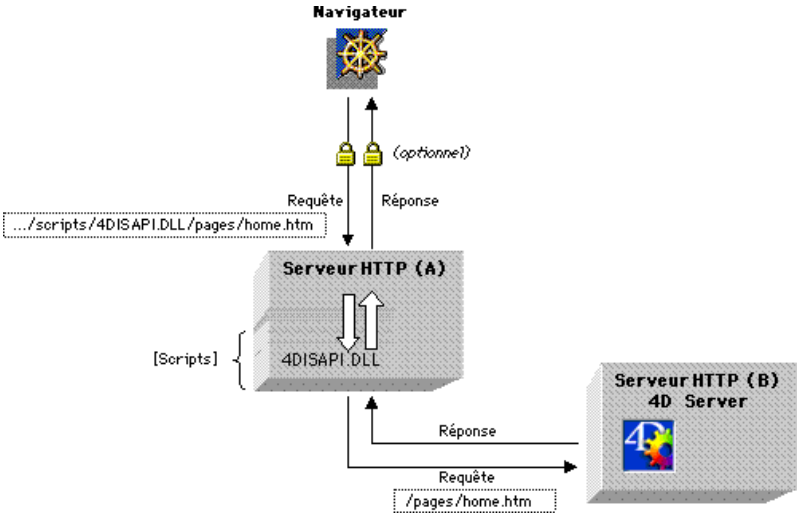
Le fonctionnement de ces deux extensions est identique. La compatibilité CGI est plus largement répandue parmi les serveurs HTTP, toutefois les performances des extensions CGI sont généralement inférieures à celles des extensions ISAPI.

Principe de fonctionnement

Le principe de fonctionnement de ces extensions est le suivant : un serveur HTTP "A" publie des pages sur Internet, et un autre serveur HTTP, "B", est par exemple un 4D Server utilisé en Intranet. Afin que les deux serveurs puissent communiquer, il vous suffit d'ajouter une extension 4DISAPI ou NPH-4DCGI dans le répertoire [Scripts] du serveur A. Lorsqu'un navigateur Web envoie une requête au serveur A, celui-ci la retransmet au serveur B via l'extension 4D ISAPI ou NPH-4DCGI, par l'intermédiaire de l'URL. La réponse est ensuite acheminée au navigateur en sens inverse. Le corps de la requête ou de la réponse HTTP n'est jamais modifié par les extensions.
La requête initiale envoyée au serveur A peut être effectuée en clair ou en mode sécurisé (SSL). La communication entre les deux serveurs HTTP et l'extension 4DISAPI.DLL s'effectue en clair.

Note : Les extensions 4DISAPI et NPH-4DCGI ne sont pas compatibles avec le mode contextuel du serveur Web 4D.

Le schéma suivant illustre ce principe :



• Les extensions reconnaissent les méthodes GET, HEAD et POST, elles gèrent les différents statuts renvoyés par 4D (*200 OK*, *302 Moved Temporarely*, *404 Not Found*...).
• Il n'est pas possible de procéder à une authentification au niveau HTTP via l'extension 4DISAPI ou NPH-CGI4D. Pour cela, il est nécessaire d'utiliser un formulaire HTML (ce qui est parfaitement envisageable en connexion sécurisée).

Note : Les extensions sont avant tout destinées à être utilisées pour recevoir et renvoyer des données dynamiques, et en particulier pour poster des données. Le simple service de pages Web n'offre pas des performances optimales en cas de transit via des extensions ISAPI ou CGI.

Installation et configuration

L'installation des extensions 4DISAPI et NPH-CGI4D s'effectue par simple copie des fichiers 4DISAPI.DLL ou NPH-CGI4D.EXE dans le dossier [Scripts] du serveur HTTP.

Chaque extension installée doit être accompagnée d'un fichier de configuration (de type *.INI). Le fichier .INI doit porter le même nom que l'extension (par exemple 4DISAPI.INI). L'extension et son fichier de configuration doivent être placés dans le même dossier.

Vous pouvez configurer un serveur HTTP de manière à ce qu'il puisse cibler plusieurs autres serveurs HTTP. Dans ce cas, placez dans le dossier [Scripts] du serveur HTTP autant d'extensions que de serveurs-cibles. Il vous suffit de les renommer (par exemple 4DISAPI2.DLL, 4DISAPI3.DLL, etc.). Veillez à insérer un fichier de configuration par extension, et à le renommer en conséquence (4DISAPI2.INI, 4DISAPI3.INI, etc.).

Le fichier .INI est constitué d'une seule section : [Forward]. Cette section accepte les commandes suivantes :

• TargetServer =
Nom ou adresse IP du serveur Web à cibler (par exemple monserveur.net ou 192.193.194.195). Laisser la chaîne vide pour repérer le serveur par son adresse si la résolution par nom n'est pas disponible
Le nom "localhost" est reconnu comme étant l'adresse 127.0.0.1.
Par défaut, l'adresse 127.0.0.1 est utilisée.

• TargetPort =
Port sur lequel le serveur-cible écoute (par exemple 81). Par défaut, le port 8080 est utilisé.

• Timeout =
Délai maximum d'attente de la réponse du serveur (exprimé en secondes). La valeur par défaut est de 30 secondes.

• Allowed =
Liste des URL autorisés, séparés par des virgules. Par exemple : /pages, /img pour n'autoriser que les URL commençant par /pages et /img. Pour autoriser la totalité du site, inscrivez une barre oblique seule / (paramétrage par défaut).

• Forbidden =
Liste des URL interdits, séparés par des virgules. Par exemple : /4dmethod, /pages2 pour interdire les URL commençant par /4dmethod et /pages2. Pour ne rien interdire, ne saisissez rien dans la liste (paramétrage par défaut).
Compte tenu des exemples d'autorisations et d'exclusions fournis ci-dessus, les URL suivants du serveur-cible seront accessibles ou non :

/pages/document.html	accessible
/pages1/onepage.html	accessible
/www/image.gif	inaccessible
/pages2/mypage.html	inaccessible
/4dmethod/myproc	inaccessible

Si un URL déclaré interdit est sollicité, l'extension retourne directement l'erreur "*HTTP/1.0 403 Forbidden*".

Utilisation

Les extensions 4DISAPI et NPH-CGI4D acceptent les URLs suivants :

• **Appel de 4D** (4D ne reçoit que la portion de chemin qui suit le nom de l'extension) :
http://adresse-serveur/cgi-bin/4disapi.dll/[chemin d'accès]
http://adresse-serveur/cgi-bin/nph-cgi4d.exe/[chemin d'accès]

• **Test de fonctionnement de l'extension** (écho de la requête) :
http://adresse-serveur/cgi-bin/4disapi.dll/~~echo
http://adresse-serveur/cgi-bin/nph-cgi4d.exe/~~echo

• **Informations sur l'extension** (support technique) :
http://adresse-serveur/cgi-bin/4disapi.dll/~~info
http://adresse-serveur/cgi-bin/nph-cgi4d.exe/~~info
Les informations suivantes sont retournées :
- nom et version de l'extension, par exemple "Script name: 4disapi.dll (6.7.0b1.2)"
- nom et version du serveur appelant l'extension, par exemple "Server software: 4D_WebStar_D/6.7"
- version du protocole HTTP, par exemple "Server protocol: HTTP/1.0"
- version du protocole CGI, par exemple "Gateway interface: CGI/1.1"

• **Test du serveur cible** (est-il joignable ?) :
http://adresse-serveur/cgi-bin/4disapi.dll/~~target
http://adresse-serveur/cgi-bin/nph-cgi4d.exe/~~target
La réponse est soit :
"Good: target server reached." : le serveur cible a répondu (quel que soit le contenu de la réponse).
soit "Bad: target server not reached." : le serveur est injoignable ou n'a pas répondu.
Dans ce cas, la raison de l'échec peut être :
- pas de fichier de configuration ;
- l'adresse ou le port cible est incorrect ;
- le serveur cible n'est pas en service ;
- le serveur a traité la requête mais est incapable de répondre.

Note sur 4D WebSTAR : 4D WebSTAR® est un des serveurs Web les plus répandus sur la plate-forme Mac OS. Diverses possibilités d'interactions directes sont possibles entre 4D WebSTAR et 4D à l'aide du plug-in 4D WebSTAR 4D Link. Pour plus d'informations sur ce plug-in, veuillez consulter la documentation de 4D WebSTAR.

Référence

FIXER EXECUTABLE CGI, FIXER VARIABLE ENVIRONNEMENT.

ARRETER SERVEUR WEB

ARRETER SERVEUR WEB

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande ARRETER SERVEUR WEB stoppe le serveur Web de l'application 4D sur laquelle elle a été exécutée (4D ou 4D Server).
Si le serveur Web était lancé, toutes les connexions Web sont interrompues et tous les process Web sont arrêtés.
Si le serveur Web n'était pas lancé, la commande ne fait rien.

Référence

LANCER SERVEUR WEB.

Connexion Web securisee

Connexion Web securisee → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai = la connexion Web est sécurisée Faux = la connexion Web n'est pas sécurisée

Description

La commande Connexion Web securisee retourne un booléen indiquant si la connexion au serveur Web 4D s'effectue en mode sécurisé via SSL (la requête débute par "https:" au lieu de "http:").

- Si la connexion est effectuée en SSL, la fonction retourne Vrai.
- Si la connexion est effectuée en mode classique (non sécurisé), la fonction retourne Faux.

Cette commande permet par exemple, le cas échéant, de refuser les tentatives de connexion en mode non sécurisé. Pour plus d'informations sur ce point, reportez-vous à la section Utiliser le protocole SSL.

Référence

GENERER DEMANDE CERTIFICAT, Utiliser le protocole SSL.

Contexte Web

Contexte Web → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai = Mode contextuel Faux = Mode sans contexte

Description

La commande Contexte Web doit être appelée depuis un process Web. Elle retourne un booléen indiquant si la connexion Web s'effectue (Vrai) ou non (Faux) en mode contextuel.

Note : La commande Contexte Web retourne toujours Faux lorsqu'elle est :
• appelée depuis un process autre qu'un process Web,
• exécutée sur un 4D Client.

L'utilisation de cette fonction est préconisée dans la Méthode base Sur connexion Web.

Exemple

Voici la structure typique de la Méthode base Sur connexion Web :

```
Si (Contexte Web)
    AvecContexte ($1;$2;$3;$4;$5;$6)
Sinon
    SansContexte ($1;$2;$3;$4;$5;$6)
Fin de si
```

Référence

INFORMATIONS PROCESS, Méthode base Sur connexion Web, Utiliser le mode contextuel.

ENVOYER BLOB HTML (blob; type{; sansContexte})

Paramètre	Type		Description
blob	BLOB	→	BLOB à envoyer au browser
type	Alpha	→	Type de données du BLOB
sansContexte	Booléen	→	Vrai = Passer en mode sans contexte, Faux ou omis = Conserver mode courant

Description

La commande ENVOYER BLOB HTML permet d'envoyer le BLOB blob au navigateur.

Le type de données contenues dans le BLOB est indiqué par le paramètre type. Ce paramètre peut contenir les valeurs suivantes :
• type = **Chaîne vide** ("") : dans ce cas, vous ne fournissez aucune information sur le BLOB. Le navigateur tentera alors d'interpréter lui-même le contenu du BLOB.
• type = **Extension de fichier** (ex. : ".HTM", ".GIF", ".JPEG", etc.) : dans ce cas, vous fournissez au navigateur, par l'intermédiaire de son extension, le type MIME des données contenues dans le BLOB. Le BLOB sera interprété en fonction de cette extension. Toutefois, l'extension doit être standard afin que le navigateur puisse l'interpréter correctement. Une liste des types MIME les plus courants et de leurs extensions est fournie ci-dessous.
• type = **Mime/Type** (ex. : "text/html", "image/tiff", etc.) : dans ce cas, vous fournissez directement au navigateur le type MIME des données contenues dans le BLOB. Cette solution est celle qui vous offre le plus de latitude. En effet, outre les types standard, vous pouvez passer un type MIME personnalisé pour envoyer des documents propriétaires en Intranet. Il vous suffit pour cela de configurer les navigateurs afin qu'ils reconnaissent le type envoyé et, par exemple, exécutent l'application correspondante. La valeur à passer dans le paramètre type est, dans ce cas "application/x-[NomDuType]"). Dans les navigateurs des postes clients, vous référencez ce type et lui associez l'action "Exécuter l'application". La commande ENVOYER BLOB HTML vous permet alors d'envoyer des documents de tout type, les clients Intranet ouvrant automatiquement l'application associée.

Note : Si le BLOB est de type "text/html" (.htm, .html, .shtm, .shtml), il est traduit et analysé comme un fichier HTML. Dans ce cas, lorsqu'elle est utilisée en mode contextuel, ENVOYER BLOB HTML fonctionne exactement comme ENVOYER FICHIER HTML. En particulier, une référence à une méthode 4D qui exécute l'instruction ENVOYER BLOB HTML("") doit être présente dans les pages reçues par le navigateur, afin de terminer l'appel à ENVOYER BLOB HTML et de retourner à l'exécution de la méthode 4D d'appel. Pour plus d'informations sur ce mécanisme, reportez-vous à la description de la commande ENVOYER FICHIER HTML.

Voici une liste des types MIME les plus courants :

Extension	Mime/Type
.htm	text/html
.html	text/html
.shtml	text/html
.shtm	text/html
.css	text/css
.pdf	application/pdf
.rtf	application/rtf
.ps	application/postscript
.eps	application/postscript
.hqx	application/mac-binhex40
.js	application/javascript
.txt	text/plain
.text	text/plain
.gif	image/gif
.jpg	image/jpeg
.jpeg	image/jpeg
.jpe	image/jpeg
.jfif	image/jpeg
.pic	image/pict
.pict	image/pict
.tif	image/tiff
.tiff	image/tiff
.mpeg	video/mpeg
.mpg	video/mpeg
.mov	video/quicktime
.moov	video/quicktime
.aif	audio/aiff
.aiff	audio/aiff
.wav	audio/wav
.ram	audio/x-pn-realaudio
.sit	application/x-stuffit
.bin	application/x-stuffit
.z	application/x-zip
.zip	application/x-zip
.gz	application/x-gzip
.tar	application/x-tar

Note : Pour plus d'informations sur ce point, veuillez consulter la section “Protocol Numbers and Assignment Services” sur le site http://www.iana.org.

Le paramètre sansContexte vous permet d’indiquer au serveur Web 4D que vous souhaitez passer du mode contextuel au mode sans contexte. Dans ce cas, passez Vrai dans le paramètre sansContexte.
Si ce paramètre est omis ou s’il contient Faux, l’envoi du BLOB est effectué dans le mode courant.

Les éventuelles références aux variables 4D et balises de type 4DSCRIPT dans la page sont toujours analysées, quel que soit le mode.

Exemple

Reportez-vous à l'exemple de la routine IMAGE VERS GIF.

Référence

ENVOYER FICHIER HTML.

ENVOYER DONNEES HTTP

ENVOYER DONNEES HTTP (données{; *})

Paramètre	Type		Description
données	BLOB	→	Données HTTP à envoyer
*	*	→	Envoi morcelé (chunked)

Description

La commande ENVOYER DONNEES HTTP permet au serveur Web 4D d'envoyer des données HTTP "brutes", pouvant être morcelées. Elle fonctionne uniquement en mode sans contexte.

Le paramètre données contient les deux parties standard d'une réponse HTTP, c'est-à-dire l'en-tête et le corps (*header* et *body*). Les données sont envoyées sans formatage préalable par le serveur. Toutefois, 4D effectue un contrôle élémentaire sur l'en-tête et le corps de la réponse afin qu'elle soit valide :
• Si l'en-tête est incomplet ou non conforme aux spécifications du protocole HTTP, 4D le modifie en conséquence.
• Si la réponse HTTP est incomplète, 4D ajoute les informations manquantes. Si, par exemple, vous souhaitez effectuer une redirection, vous devez écrire :

```
HTTP/1.1 302
Location : http://...
```

Si vous passez uniquement :

```
Location : http://...
```

4D complétera la réponse en ajoutant HTTP/1.1 302.

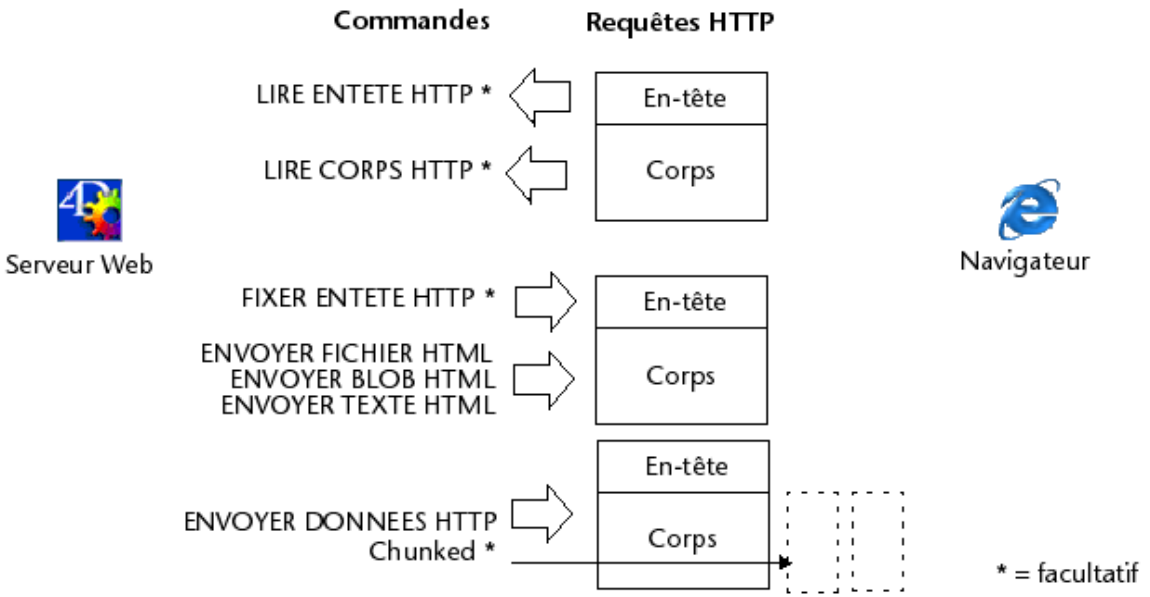
Le paramètre optionnel * permet de déclarer que la réponse sera envoyée sous forme "morcelée" (*chunked*). Le découpage des réponses peut être utile lorsque le serveur envoie une réponse sans connaître sa longueur totale (par exemple si la réponse n'a pas encore été générée). Tous les navigateurs compatibles HTTP/1.1 acceptent les réponses "morcelées". Si vous passez le paramètre *, le serveur Web inclura automatiquement le champ transfer-encoding: chunked dans l'en-tête de la réponse, si nécessaire (vous pouvez gérer manuellement l'en-tête de la réponse si vous le souhaitez). Le reste de la réponse sera également formaté en respectant la syntaxe de l'option *chunked*. Les réponses morcelées comportent un seul en-tête et un nombre indéfini de corps.
Toutes les instructions ENVOYER DONNEES HTTP suivant l'exécution de ENVOYER DONNEES HTTP(données;*) au sein de la même méthode seront considérées comme partie de la réponse (qu'elles contiennent ou non le paramètre *). Le serveur met un terme à l'envoi morcelé à la fin de l'exécution de la méthode.

Note : Si le client Web ne prend pas en charge le protocole HTTP/1.1, 4D convertira automatiquement la réponse au format compatible HTTP/1.0 (l'envoi ne sera pas morcelé).

Dans ce cas toutefois, il est possible que le résultat ne corresponde pas à vos attentes. Il est donc recommandé de tester si le navigateur est compatible HTTP/1.1 et d'envoyer une réponse adaptée. Pour cela, vous pouvez utiliser une méthode de ce type :

```
C_BOOLEEN($0)
TABLEAU TEXTE(tabChamps;0)
TABLEAU TEXTE(tabValeurs;0)
LIRE ENTETE HTTP(tabChamps;tabValeurs)
$0:=Faux
Si (Taille tableau(tabValeurs)>=3)
   Si (Position("HTTP/1.1";tabValeurs{3})>0)
      $0:=Vrai   ` Le navigateur est compatible HTTP/1.1, on retourne Vrai dans $0
   Fin de si
Fin de si
```

Combinée à la commande LIRE CORPS HTTP et aux autres commandes du thème "Serveur Web", cette commande complète la gamme d'outils mis à la disposition des développeurs 4D pour traiter de manière entièrement personnalisée les connexions HTTP entrantes et sortantes. Ces différents outils sont présentés dans le schéma suivant :



Exemple

Cet exemple illustre l'emploi de l'option *chunked* avec la commande ENVOYER DONNEES HTTP. Les données (une suite de chiffres) sont envoyées en 100 morceaux générés à la volée dans une boucle. A noter que l'en-tête de la réponse n'est pas explicitement défini : la commande ENVOYER DONNEES HTTP l'enverra automatiquement et y insérera le champ transfer-encoding: chunked car le paramètre * est utilisé.

```
C_ENTIER LONG($cpt)
C_BLOB($mon_blob)
C_TEXTE($output)

Boucle ($cpt;1;100)
    $output:="["+Chaine($cpt)+"]"
    TEXTE VERS BLOB($output;$mon_blob;UTF8 Texte sans longueur)
    ENVOYER DONNEES HTTP($mon_blob;*)
Fin de boucle
```

Référence

LIRE CORPS HTTP, LIRE ENTETE HTTP.

ENVOYER FICHIER HTML (fichierHTML)

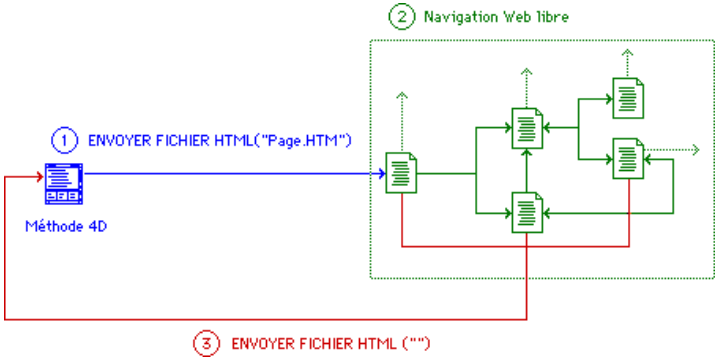
Paramètre	Type		Description
fichierHTML	Alpha	-->	Chemin d'accès HTML au fichier HTML ou Chaîne vide pour terminer ENVOYER FICHIER HTML en mode contextuel

Description

La commande ENVOYER FICHIER HTML envoie au navigateur Web la page Web stockée dans le document HTML dont vous passez le chemin d'accès dans fichierHTML.

Par défaut, 4D recherche le document HTML à l'intérieur du dossier racine HTML, défini par défaut dans les Préférences de l'application.
Cette commande accepte en paramètre un chemin d'accès exprimé en syntaxe HTML uniquement : les noms de répertoires ou de dossiers doivent être séparés par une barre oblique ("/"), quelle que soit la plate-forme que vous utilisez.
Si vous passez un chemin d'accès HTML invalide, une erreur liée à la gestion de fichiers de votre système d'exploitation est générée. Vous pouvez intercepter l'erreur à l'aide d'une méthode installée par la commande APPELER SUR ERREUR. Si la méthode affiche une boîte de dialogue d'alerte ou de message, celle-ci apparaît sur le poste du navigateur.

La syntaxe ENVOYER FICHIER HTML("") (vous passez une chaîne vide dans le paramètre fichierHTML) permet, en mode contextuel, d'achever l'appel à la commande ENVOYER FICHIER HTML ayant démarré le mode HTML. Ce principe est illustré dans le schéma suivant :



(1) En mode contextuel, une méthode 4D (projet, objet ou base) exécute un ENVOYER FICHIER HTML, envoyant un document HTML au navigateur.

(2) La page Web initiale envoyée au navigateur peut comporter des liens HTML vers d'autres pages Web ou peut référencer elle-même des méthodes 4D appelant ENVOYER FICHIER HTML pour envoyer d'autres pages Web. Ces autres pages peuvent également comporter des liens ou référencer des méthodes 4D pour permettre d'accéder à d'autres pages, etc. Pendant que vous naviguez parmi les pages Web, vous pouvez également utiliser les outils de navigation du navigateur, comme le bouton Précédent.

(3) N'importe laquelle des pages Web peut contenir une référence à une méthode 4D qui exécute l'instruction ENVOYER FICHIER HTML(""). Cette instruction termine l'appel à ENVOYER FICHIER HTML qui a lancé le processus et permet de retourner à l'exécution de la méthode 4D ayant initialement démarré la navigation Web libre.

Une fois que l'instruction ENVOYER FICHIER HTML a été exécutée, la variable système OK est mise à jour : si le fichier à envoyer existe et si le timeout n’est pas dépassé, la variable OK prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

Note : Si vous appelez ENVOYER FICHIER HTML depuis un process qui n'est pas un process Web, la commande ne fait rien. Aucune erreur n'est retournée, l'appel est simplement ignoré.

Les éventuelles références aux variables 4D et aux balises de type 4DSCRIPT dans la page sont toujours analysées, quel que soit le mode.

Exemples

(1) Le dossier racine HTML de la base est le dossier WebDocs. Il contient les éléments suivants :

```
..\WebDocs\HTM\MaPage.HTM
```

L'envoi de la page Web "MaPage.HTM" doit être effectué de cette manière :

```
ENVOYER FICHIER HTML ("HTM/MaPage.HTM")
```

(2) Exemple en mode contextuel : pendant une session Web 4D, vous ajoutez des enregistrements à l'aide d'un formulaire 4D. Dans ce formulaire se trouve un bouton bAide dont la méthode objet est la suivante :

```
   ` Méthode objet du bouton bAide
ENVOYER FICHIER HTML ("Aide.HTM")
```

A partir du document Aide.HTM, vous pouvez librement naviguer parmi différentes pages HTML qui implémentent un système d'aide en ligne pour votre site Web. Dans chaque page, vous placez un bouton "submit" dont le titre est Fin, permettant de retourner à la saisie de données. Pour cela, chaque document HTML doit comporter cette définition pour le bouton "submit" :

```
<!—bTerminé bouton submit—>
<P><INPUT TYPE="submit" NAME="bFin" VALUE="Done"></P>
```

ainsi que cette définition de l'action FORM POST :

```
<!— Execution de la méthode 4D htm_Aide_Fin si clic sur un bouton submit—>
<FORM action="/4DMETHOD/htm_Aide_Fin" method="POST">
```

Du côté de 4D, la méthode projet htm_Aide_Fin termine l'appel à ENVOYER FICHIER HTML, initié par le bouton bAide :

```
` Méthode projet htm_Aide_Fin
ENVOYER FICHIER HTML ("")
```

L'appel à ENVOYER FICHIER HTML est la dernière ligne de la méthode objet du bouton bAide ; la méthode s'achève alors et vous retournez à la saisie de données.

Variables et ensembles système

Si le fichier à envoyer existe et si le timeout n’est pas dépassé, la variable OK prend la valeur 1. Dans le cas contraire, elle prend la valeur 0.

Référence

Associer des objets 4D à des objets HTML, ENVOYER BLOB HTML, Premiers pas.

ENVOYER REDIRECTION HTTP (url{; *})

Paramètre	Type		Description
url	Alpha	→	Nouvel URL
*	*	→	Si spécifié = l'URL n'est pas traduit, Si omis = l'URL est traduit

Description

La commande ENVOYER REDIRECTION HTTP permet de transformer un URL en un autre.

Le paramètre url contient le nouvel URL qui permet de rediriger la requête. Si ce paramètre est un url vers un fichier, il doit contenir la référence à ce fichier, par exemple : ENVOYER REDIRECTION HTTP ("/MaPage.HTM").

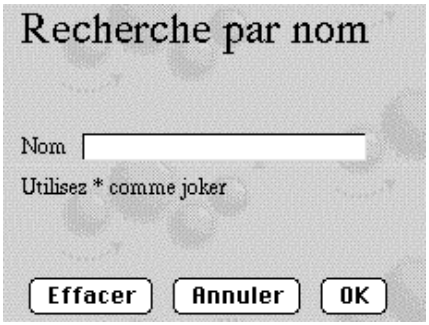
Lorsque cette commande est appelée depuis le mode contextuel, le process Web est tué juste après son exécution. Elle prévaut sur les commandes d'envoi de données (ENVOYER FICHIER HTML, ENVOYER BLOB HTML, etc.) éventuellement placées dans la même méthode.

Cette commande permet également de rediriger une requête vers un autre serveur Web.

4D encode automatiquement les caractères spéciaux de l'URL. Si vous passez le caractère *, 4D ne traduira pas les caractères spéciaux de l'URL.

Exemple

Vous pouvez utiliser cet URL pour effectuer, à l'aide de pages statiques, des recherches personnalisées dans 4D. Imaginez que vous ayez placé dans une page HTML statique les éléments suivants :



L'action POST "/4dcgi/rech" a été associée à la zone de texte et aux boutons **OK** et **Annuler**.

Dans la partie (ou la sous-méthode) de la Méthode base Sur connexion Web gérant le mode sans contexte, placez les instructions suivantes :

```
Au cas ou
    : ($1="/4dcgi/rech")   `Lorsque 4D reçoit cet URL
            `Si le bouton OK a été utilisé et le champ 'nom' contient une valeur
        Si ((bOK="OK") & (nom # ""))
                `Transformer l'URL afin d'exécuter le code de la recherche, placé plus
                `loin dans la même méthode
            ENVOYER REDIRECTION HTTP("/4dcgi/rech?"+nom)
        Sinon   `Sinon retourner à la page de départ
            ENVOYER REDIRECTION HTTP("/page1.htm")
        Fin de si
        ...

    : ($1="/4dcgi/rech?@")   `Si l'URL a été redirigé
        ... `Placez ici le code de la recherche
Fin de cas
```

ENVOYER TEXTE HTML (texteHTML{; sansContexte})

Paramètre	Type		Description
texteHTML	Texte	→	Champ ou variable texte au format HTML à envoyer au navigateur
sansContexte	Booléen	→	Vrai = Passage en mode sans contexte Faux ou omis = Conserver mode courant

Description

La commande ENVOYER TEXTE HTML permet d'envoyer directement des données texte formatées en HTML.

Le paramètre texteHTML contient les données à envoyer. 4D n'effectue aucun contrôle sur le contenu de ce paramètre, vous devez donc veiller à ce que le codage HTML soit correct. Le texte doit être encodé en ISO Latin-1.

Note : Cette commande équivaut strictement à l'envoi d'un BLOB ayant le type "text/html" à l'aide de la commande ENVOYER BLOB HTML.

Le paramètre sansContexte vous permet d'indiquer au serveur Web 4D que vous souhaitez passer du mode "contextuel" au mode "sans contexte". Dans ce cas, passez Vrai dans sansContexte.
Pour conserver le mode courant, passez Faux dans sansContexte ou omettez ce paramètre.

Les éventuelles références aux variables 4D et balises de type 4DSCRIPT dans le texte sont toujours analysées, quel que soit le mode.

Exemple

La méthode suivante :

```
TEXTE VERS BLOB("<html><head></head><body>"+Chaine(Heure courante)+
                                    "</body></html>";$blob;UTF8 Texte sans longueur)
ENVOYER BLOB HTML ($blob;"text/html")
```

... peut être remplacée par :

```
ENVOYER TEXTE HTML ("<html><head></head><body>"+Chaine(Heure courante)+
                                                    "</body></html>")
```

Référence

ENVOYER BLOB HTML, Mac vers ISO.

FIXER ENTETE HTTP (entête|tabChamps{; tabValeurs})

Paramètre	Type		Description
entête|tabChamps	Texte|Tab Texte	→	Champ ou variable contenant l'en-tête HTTP de la requête ou Tableau des champs de l'en-tête HTTP
tabValeurs	Tab Texte	→	Contenu des champs de l'en-tête HTTP

Description

La commande FIXER ENTETE HTTP permet de fixer les champs de l'en-tête HTTP de la réponse faite au navigateur Web par 4D. Elle n'a d'effet que dans un process Web en mode sans contexte.
Cette commande vous permet, en particulier, de gérer des "cookies".

FIXER ENTETE HTTP admet deux syntaxes :

• **Première syntaxe :** FIXER ENTETE HTTP **(entête)**
Vous passez dans le paramètre entête, de type variable ou champ texte, les champs de l'en-tête HTTP que vous souhaitez fixer. Les champs doivent être séparés entre eux par un retour chariot ou une séquence cr/lf (retour chariot/retour à la ligne), sous Windows et Mac OS, 4D se charge du formatage de la réponse.

Voici un exemple de "cookie" personnalisé :

```
C_TEXTE($vTcookie)
$vTcookie:="SET-COOKIE: USER="+Chaine(Abs(Hasard))+"; PATH=/"
FIXER ENTETE HTTP($vTcookie)
```

Note : Il n'est pas possible de passer une constante texte littérale directement dans le paramètre entête. Vous devez utiliser une variable ou un champ intermédiaire.

Pour plus d'informations sur la syntaxe à appliquer dans les en-têtes HTTP, veuillez consulter sur Internet les R.F.C (Request For Comments) à l'adresse http://www.w3c.org.

• **Deuxième syntaxe :** FIXER ENTETE HTTP **(tabChamps; tabValeurs)**
L'en-tête HTTP est défini à l'aide de deux tableaux texte, tabChamps et tabValeurs. L'en-tête sera écrit de la manière suivante :

```
tabChamps{1}:="X-VERSION"
tabChamps{2}:="X-STATUS"
tabChamps{3}:="Set-Cookie"

tabValeurs{1}:="HTTP/1.0" *
tabValeurs{2}:="200 OK" *
tabValeurs{3}:="C=HELLO"
```

***** Ces deux premiers éléments constituent la première ligne de la réponse. Lorsqu'ils sont saisis, ils doivent impérativement être les éléments 1 et 2 des tableaux. Il est toutefois possible de les omettre et d'écrire seulement — 4D se chargeant de formater l'en-tête :

```
tabChamps{1}:="Set-Cookie"
tabValeurs{1}:="C=HELLO"
```

Si vous ne spécifiez pas de statut, celui-ci est automatiquement HTTP/1.0 200 OK. Conformément à la norme HTTP, les noms des champs HTTP sont toujours libellés en anglais.

Les champs **Content-Length**, **Server** et **Date** sont toujours fixés par 4D.

Référence

LIRE ENTETE HTTP.

FIXER EXECUTABLE CGI

FIXER EXECUTABLE CGI (url1{; url2})

Paramètre	Type		Description
url1	Chaîne	→	URL d'accès
url2	Chaîne	→	URL d'accès

Description

La commande FIXER EXECUTABLE CGI permet d'exécuter un CGI sans que celui-ci soit visible pour l'utilisateur Web dans l'URL. Cette commande peut notamment être utilisée dans la Méthode base Sur authentification Web, afin de déterminer par exemple le CGI à exécuter. Elle fonctionne sous Mac OS X et sous Windows.

Passez dans le paramètre url1 l'URL d'accès au CGI à exécuter. Par exemple, si vous écrivez FIXER EXECUTABLE CGI("/myfile.pl"), le serveur Web 4D exécutera le CGI myfile.pl — cette application doit se trouver dans le dossier racine par défaut du serveur Web.
Si vous passez une chaîne vide ("") dans url1, 4D exécutera directement le CGI défini dans l'URL envoyé par le navigateur, le cas échéant.

Passez dans le paramètre facultatif url2 l'URL d'accès à un fichier devant être traité par le CGI. Par exemple, si vous écrivez FIXER EXECUTABLE CGI("cgi-bin/Perl2.cgi";"Perl2.pl"), le serveur Web exécutera le CGI Perl2.cgi (situé dans le dossier cgi-bin) en lui passant le fichier Perl2.pl. Si vous passez une chaîne vide ("") dans url2, 4D passera au CGI pour traitement le fichier défini dans l'URL envoyé par le navigateur. Ce mécanisme est utilisé notamment par PHP. Exemple : FIXER EXECUTABLE CGI("/cgi-bin/php";"").

Si l'URL d'accès spécifié par la commande est incorrect, le navigateur affichera la page d'erreur "Fichier non trouvé".
A noter que la commande FIXER EXECUTABLE CGI ne retourne pas directement d'erreur. Cette commande définit uniquement une "valeur courante" qui est utilisée ultérieurement, lorsque le CGI sera appelé. En cas d'appels multiples de cette commande, seule la valeur définie par le dernier appel sera utilisée.

Exemple

Dans cet exemple, le fichier example.php, non situé dans le dossier cgi-bin, est traité par le CGI Perl2.cgi, situé dans ce dossier :

```
FIXER EXECUTABLE CGI("/cgi-bin/Perl2.cgi";"example.php")
```

Référence

Support des CGI.

FIXER LIMITES AFFICHAGE WEB

FIXER LIMITES AFFICHAGE WEB (nombreEnr{; nombrePages{; imageRéf}})

Paramètre	Type		Description
nombreEnr	Numérique	→	Nombre maximum d'enregistrements à afficher dans chaque page HTML
nombrePages	Numérique	→	Nombre maximum de références de pages en bas de chaque page HTML
imageRéf	Numérique	→	Numéro de référence d'image pour l'icône du bouton d'affichage pleine page d'un enregistrement

Description

La commande FIXER LIMITES AFFICHAGE WEB modifie la manière dont 4D affiche une sélection d'enregistrements dans un navigateur Web lorsque vous utilisez les commandes VISUALISER SELECTION ou MODIFIER SELECTION. Cette commande fonctionne en mode contextuel uniquement.

Lorsque vous affichez une sélection d'enregistrements avec 4D, le programme ne charge pas tous les enregistrements de la sélection, mais uniquement ceux qui sont visibles dans la fenêtre. Ainsi, même si des milliers d'enregistrements sont sélectionnés, leur affichage reste rapide : seuls les enregistrements visibles simultanément sont chargés du disque. Ensuite, si vous faites défiler la liste d'enregistrements ou redimensionnez la fenêtre, 4D charge en conséquence les enregistrements.

Avec le Web, 4D découpe en pages la sélection d'enregistrements à afficher. En fait, sans système de pagination, l'affichage d'une sélection de milliers d'enregistrements signifierait la circulation de milliers d'enregistrements sur votre réseau Intranet ou sur Internet et leur affichage dans une seule page Web. Non seulement cela prendrait un certain temps pour télécharger tous les enregistrements mais, de plus, votre navigateur Web serait rapidement à court de mémoire.

Par défaut, 4D affiche les 20 premiers enregistrements (s'ils existent) de la sélection et insère à la fin de chaque page HTML 20 liens vers les 20 premières pages (si elles existent) de la sélection. Cela signifie que par défaut, vous pouvez accéder aux 400 premiers enregistrements (s'ils existent) de la sélection en cliquant sur le lien de page placé à la fin de chaque page de la sélection. Notez que ce système de pagination ne nécessite aucune ligne de code, il se met automatiquement en place lors de l'exécution des commandes VISUALISER SELECTION ou MODIFIER SELECTION.

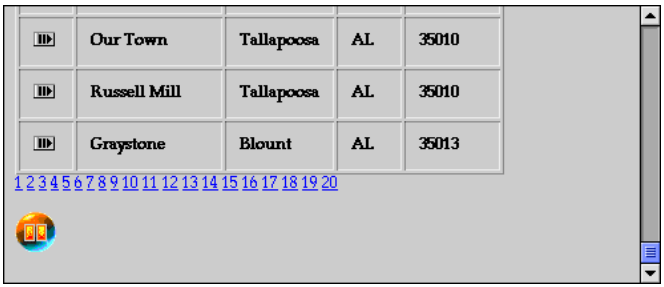
FIXER LIMITES AFFICHAGE WEB vous permet de modifier ces paramétrages. Dans nombreEnr, vous indiquez le nombre maximum d'enregistrements qui doivent être affichés par page de la sélection. Dans nombrePages, vous indiquez le nombre maximum de liens vers d'autres pages de la sélection qui doivent être affichés en bas de chaque page de la sélection.

Si, par exemple, vous disposez d'une sélection de 10 000 enregistrements et voulez pouvoir naviguer parmi la totalité de ces enregistrements dans une seule sélection, vous pouvez passer 100 dans les paramètres nombreEnr et nombrePages. Rappelez-vous toutefois que les données vont circuler sur le réseau ou sur Internet ; en particulier dans le cas d'Internet, la rapidité est un facteur à surveiller en cas de modification des paramètres d'affichage de la sélection.

Enfin, FIXER LIMITES AFFICHAGE WEB vous permet de remplacer l'icône par défaut du bouton d'affichage d'un enregistrement en pleine page. Pour cela, passez dans le paramètre imageRéf le numéro de référence de l'image, stockée dans la librairie d'images de la base, que vous souhaitez utiliser comme nouvelle icône.
FIXER LIMITES AFFICHAGE WEB n'affecte que les appels ultérieurs aux commandes VISUALISER SELECTION et MODIFIER SELECTION et sa portée est locale au process courant.

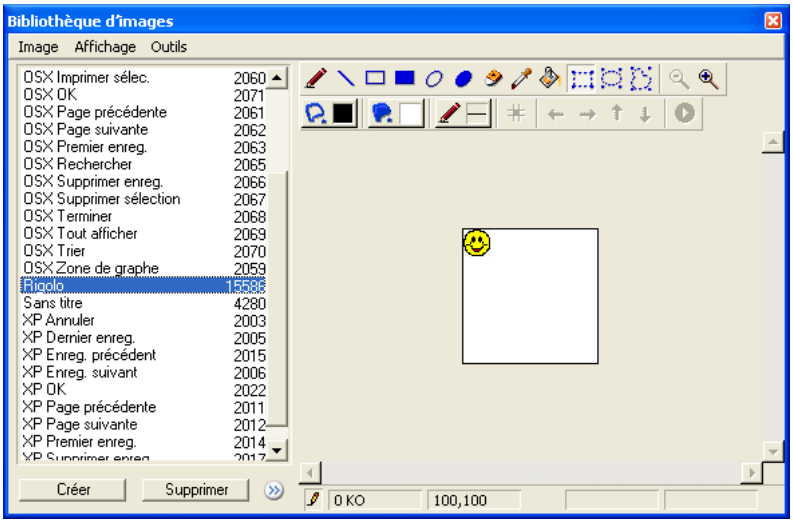
Exemple

Dans l'exemple suivant, un VISUALISER SELECTION ou un MODIFIER SELECTION est exécuté pour la table [Codes américains]. Au niveau du navigateur Web, 4D affiche par défaut les enregistrements de la manière suivante :



Notez que vous pouvez naviguer parmi les 400 premiers enregistrements.

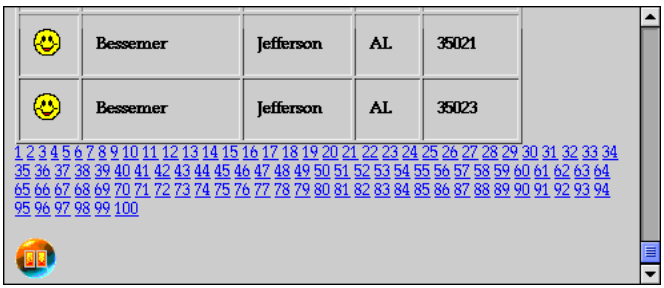
Si l'image suivante est placée dans la bibliothèque d'images de la base :



Si, également, la méthode projet qui affiche la sélection exécute l'instruction suivante avant d'appeler VISUALISER SELECTION ou MODIFIER SELECTION :

FIXER LIMITES AFFICHAGE WEB (50;100;15586)

Alors, au niveau du navigateur Web, la sélection des enregistrements apparaîtra ainsi :



Vous pouvez donc naviguer parmi les 5 000 premiers enregistrements de la sélection.

Référence

MODIFIER SELECTION, Utiliser le mode contextuel, VISUALISER SELECTION.

FIXER PAGE ACCUEIL (homePage)

Paramètre	Type		Description
homePage	Alpha	→	Nom de page ou chemin d'accès HTML à la page ou "" pour ne pas envoyer de page d'accueil personnalisée

Description

La commande FIXER PAGE ACCUEIL vous permet de modifier la page d'accueil (page Home) personnalisée pour le process Web courant.

La page définie est liée au process Web, vous pouvez donc définir des pages d'accueil différentes en fonction, par exemple, de l'utilisateur connecté. Cette page peut être statique ou semi-dynamique.

Vous passez dans le paramètre homePage le nom de la page HTML d'accueil ou le chemin d'accès HTML complet à la page.

Pour ne plus envoyer homePage comme page d'accueil pour le process Web courant, appelez de nouveau la commande FIXER PAGE ACCUEIL en passant une chaîne vide ("") dans homePage.

Note : Vous pouvez également définir une page d'accueil par défaut dans les Préférences de l'application. Dans ce cas, la page s'applique par défaut à toutes les connexions Web, quel que soit le mode de démarrage (contextuel ou sans contexte) du serveur Web.

Référence

Paramétrages du serveur Web.

FIXER RACINE HTML

version 11 (Modifiée)

FIXER RACINE HTML (dossierRacine)

Paramètre	Type		Description
dossierRacine	Chaîne	→	Chemin d'accès du dossier racine du serveur Web

Description

La commande FIXER RACINE HTML permet de modifier le dossier racine par défaut dans lequel 4D ira rechercher les fichiers HTML demandés au serveur Web.

Cette commande ne tient pas compte du dossier racine HTML par défaut éventuellement défini dans les Préférences de la base. Pour plus d'informations sur ce dossier, reportez-vous à la section Sécurité des connexions.

L'emplacement du dossier racine peut être exprimé soit en syntaxe HTML (type URL), soit en syntaxe système (chemin absolu) :
• Syntaxe HTML : les noms de dossiers sont séparés par une barre oblique ("/"), quelle que soit la plate-forme que vous utilisez.
• Syntaxe système : chemin d'accès absolu ("nom long") respectant la syntaxe de la plate-forme courante, par exemple :
- (Mac OS) Disque:Applications:monserv:dossier
- (Windows) C:\Applications\monserv\dossier

Notes :
• La prise en compte du nouveau dossier racine nécessite le redémarrage du serveur Web.
• Vous pouvez connaître à tout moment l'emplacement du dossier racine courant à l'aide de la commande Dossier 4D.

Si vous passez un chemin d'accès invalide, une erreur liée à la gestion de fichiers du système d'exploitation est générée. Vous pouvez intercepter l'erreur à l'aide d'une méthode installée par la commande APPELER SUR ERREUR. Si la méthode affiche une boîte de dialogue d'alerte ou de message, celle-ci apparaît sur l'écran du navigateur.

Référence

APPELER SUR ERREUR, Dossier 4D.

FIXER TEMPORISATION WEB (timeout)

Paramètre	Type		Description
timeout	Numérique	→	Délai de fermeture de la connexion Web exprimé en secondes

Description

La commande FIXER TEMPORISATION WEB définit le timeout (délai d'attente maximum avant la déconnexion automatique du client Web) pour les process de connexion Web en mode contextuel. La valeur du timeout par défaut est de 5 minutes.

Vous pouvez augmenter ou réduire cette valeur en passant dans le paramètre timeout une nouvelle valeur, exprimée en secondes.

La commande prend effet immédiatement et a pour portée la session de travail.
Si la commande FIXER TEMPORISATION WEB est appelée depuis un process Web, la valeur de timeout ne s'applique qu'à ce process.
Si la commande est appelée en-dehors d'un process Web, la valeur de timeout s'applique à tous les process Web créés par la suite.

Référence

Paramétrages du serveur Web, Utiliser le mode contextuel.

LANCER SERVEUR WEB

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande LANCER SERVEUR WEB démarre le serveur Web de l'application 4D sur laquelle elle a été exécutée (4D ou 4D Server). La base est alors publiée sur votre réseau Intranet ou sur Internet.

Si le serveur Web a été correctement lancé, la variable système OK prend la valeur 1, sinon — si par exemple le protocole réseau TCP/IP n'est pas correctement configuré — OK prend la valeur 0 (zéro).

Référence

ARRETER SERVEUR WEB.

Ensembles et variables système

Si le serveur Web est correctement démarré, OK prend la valeur 1, sinon OK prend la valeur 0 (zéro).

LIRE CORPS HTTP (corps)

Paramètre	Type	Description
corps	BLOB | Texte←	Champ corps (Body) de la requête HTTP

Description

La commande LIRE CORPS HTTP retourne le corps (body) de la requête HTTP en cours de traitement. Le corps HTTP est retourné tel quel, sans traitement ni analyse.

Cette commande fonctionne uniquement en mode sans contexte. Elle peut être appelée depuis la Méthode base Sur authentification Web, la Méthode base Sur connexion Web ou toute méthode Web exécutée en mode sans contexte.

Vous pouvez passer dans le paramètre corps une variable ou un champ de type BLOB ou Texte. A noter que si votre base fonctionne en mode compatibilité ASCII, le type BLOB est préférable car il n'est pas limité en nombre de caractères. Le type Texte, quant à lui, est limité à 2Go 32 000 caractères ; en cas de dépassement, les données reçues seront tronquées.
Si votre base fonctionne en mode standard (Unicode), le type Texte sera généralement suffisant (le paramètre corps peut recevoir jusqu'à 2 Go de texte).

Cette commande permet par exemple d'effectuer des recherches dans le corps des requêtes. Elle permet également aux utilisateurs avancés de mettre en place un serveur WebDAV au sein d'une base 4D.

Exemple

Dans cet exemple, une requête simple est envoyée au serveur Web de 4D et le contenu du champ HTTP corps est visualisé dans le débogueur. Voici le formulaire envoyé au serveur Web de 4D, ainsi que le code HTML correspondant :

Formulaire

Corps

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org
<html>
        <head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
        <title>Page de test</title>
        </head>

        <body>
        TEST
        <FORM ACTION="/4DACTION/Test4D2004" METHOD=POST>
        <br>Nom</br>
        <input type="text" value="Saisissez votre nom" name="vnom">
        <br>
        <input type="submit">
        </FORM>
        </body>

</html>
```

Voici la méthode *Test4D2004* :

```
C_BLOB($requete)
C_TEXTE($texteRequete)

LIRE CORPS HTTP($requete)
$texteRequete:=BLOB vers texte($requete;UTF8 Texte sans longueur)
ENVOYER FICHIER HTML("page.html")
```

Note : Cette méthode a été déclarée “Disponible via 4DACTION, 4DMETHOD et 4DSCRIPT” dans ses propriétés.

Lorsque le formulaire est soumis au serveur Web, la variable *$texteRequete* reçoit le texte du champ body de la requête HTTP, soit "vnom=Dupont".

Référence

LIRE ENTETE HTTP.

LIRE ENTETE HTTP (entête|tabChamps{; tabValeurs})

Paramètre	Type		Description
entête|tabChamps	Texte|Tab Texte	←	En-tête HTTP de la requête ou Champs de l'en-tête HTTP
tabValeurs	Tab Texte	←	Contenu des champs de l'en-tête HTTP

Description

La commande LIRE ENTETE HTTP retourne, soit sous forme de chaîne, soit sous forme de deux tableaux, l'en-tête HTTP de la requête en cours de traitement.

Cette commande fonctionne uniquement en mode sans contexte. Elle peut être appelée depuis toute méthode (Méthode base Sur connexion Web, Méthode base Sur authentification Web, méthode appelée par "/4DACTION"...) exécutée dans un process Web en mode sans contexte.
Appelée en mode contextuel, LIRE ENTETE HTTP retourne des chaînes vides.

• Première syntaxe : LIRE ENTETE HTTP (entête)
Lorsque vous utilisez cette syntaxe, le résultat retourné dans la variable entête est du type suivant :

"GET /page.html HTTP\1.0"+Caractere(13)+Caractere(10)+"User-Agent: browser"+Caractere(13)+Caractere(10)+"Cookie: C=HELLO"

Chaque champ d'en-tête est séparé par une séquence CR+LF (*Retour chariot+Retour à la ligne*), sous Windows et Mac OS.

• Seconde syntaxe : LIRE ENTETE HTTP (tabChamps; tabValeurs)
Lorsque vous utilisez cette syntaxe, les résultats retournés dans les tableaux tabChamps et tabValeurs sont du type suivant :

tabChamps{1} = "X-METHOD"	tabValeurs{1} = "GET" *
tabChamps{2} = "X-URL"	tabValeurs{2} = "/page.html" *
tabChamps{3} = "X-VERSION"	tabValeurs{3} = "HTTP/1.0" *
tabChamps{4} = "User-Agent"	tabValeurs{4} = "browser"
tabChamps{5} = "Cookie"	tabValeurs{5} = "C=HELLO"

* Ces trois premiers éléments ne correspondent pas à des champs HTTP. Ils constituent la première ligne de la requête.

Conformément à la norme HTTP, les noms des champs sont toujours libellés en anglais.

A titre indicatif, voici une liste non exhaustive des champs HTTP pouvant être présents dans une requête :
• Accept : ce que le navigateur est susceptible d'accepter comme contenu.
• Accept-Language : la ou les langue(s) acceptée(s) par le navigateur (pour information). Permet de choisir une page d'accueil en fonction de la langue préférée du navigateur.
• Cookie : liste des cookies.
• From : adresse e-mail de l'utilisateur du navigateur.
• Host : nom ou adresse du serveur (par exemple, dans le cas de l'URL http://monserveurweb/mapage.html, Host prend la valeur "monserveurweb"). Permet de gérer les cas où plusieurs noms pointent vers la même adresse IP (virtual hosting).
• Referer : provenance de la requête (par exemple http://monserveurweb/mapage1.html), c'est-à-dire la page que l'utilisateur affiche s'il clique sur le bouton **Précédent** de son navigateur.
• User-Agent : nom et version du navigateur ou du proxy.

Exemple

• Cette méthode permet de récupérer le contenu de tout champ d'en-tête de requête HTTP :

```
    ` Méthode projet GetHTTPField
    ` GetHTTPField ( Texte ) -> Texte
    ` GetHTTPField ( Nom en-tête HTTP ) -> Contenu en-tête HTTP
C_TEXTE($0;$1)
C_ENTIER LONG($vlElem)
TABLEAU TEXTE($noms;0)
TABLEAU TEXTE($valeurs;0)
$0:=""
LIRE ENTETE HTTP ($noms;$valeurs)
$vlElem:=Chercher dans tableau($noms;$1)
Si ($vlElem>0)
   $0:=$valeurs{$vlElem}
Fin de si
```

• Une fois que cette méthode projet est écrite, vous pouvez l'appeler ainsi :

```
    ` Contenu de l'en-tête Cookie
$cookie:=GetHTTPField("Cookie")
```

• Vous pouvez également envoyer des pages différentes en fonction de la langue du navigateur (par exemple dans la Méthode base Sur connexion Web) :

```
$langue:=GetHTTPField("Accept-Language")
Au cas ou
    :($langue="@fr@")    `Français (cf. liste ISO 639)
        ENVOYER FICHIER HTML("index_fr.html")
    :($langue="@es@")    `Espagnol (cf. liste ISO 639)
        ENVOYER FICHIER HTML("index_es.html")
    Sinon
        ENVOYER FICHIER HTML("index.html")
Fin de cas
```

Note : Les navigateurs Web permettent de définir plusieurs langues par défaut. Elles sont listées dans le champ “Accept-Language”, séparées par des ";". Leur priorité est définie par leur position au sein de la chaîne ; il peut donc être utile de tester la position des langues dans la chaîne.

• Exemple de gestion des hôtes virtuels (par exemple dans la Méthode base Sur connexion Web). Les trois noms “home_site.com”, “home_site1.com” et “home_site2.com” pointent vers la même adresse IP, par exemple 192.1.2.3.

```
$host:=GetHTTPField("Host")
Au cas ou
    :($host="www.site1.com")
        ENVOYER FICHIER HTML("home_site1.com")
    :($host="www.site2.com")
        ENVOYER FICHIER HTML("home_site2.com")
    Sinon
        ENVOYER FICHIER HTML("home_site.com")
Fin de cas
```

Référence

FIXER ENTETE HTTP, LIRE CORPS HTTP.

LIRE VARIABLES FORMULAIRE WEB (tabNoms; tabValeurs)

Paramètre	Type		Description
tabNoms	Tableau Texte	←	Noms des variables du formulaire Web
tabValeurs	Tableau Texte	←	Valeurs des variables du formulaire Web

Description

La commande LIRE VARIABLES FORMULAIRE WEB remplit les tableaux texte tabNoms et tabValeurs avec, respectivement, les noms et les valeurs des variables contenues dans un formulaire Web "soumis" (c'est-à-dire envoyé) au serveur Web.
Cette commande récupère la valeur de toutes les variables pouvant être incluses dans des pages HTML : zones de saisie, boutons, cases à cocher, boutons radio, pop up menus, listes d'options...

Note : Dans le cas des cases à cocher, le nom de la variable et sa valeur ne sont retournés que si la case est effectivement cochée.

La commande fonctionne en mode sans contexte ou en mode contextuel, quel que soit le type d'URL ou de formulaire (méthode POST ou GET) envoyé au serveur Web.
Cette commande peut être appelée, selon les besoins, dans la Méthode base Sur connexion Web ou toute autre méthode 4D qui résulte de la soumission d'un formulaire.

Précisions sur les formulaires Web et les actions associées
Un formulaire est composé de "zones de saisie" (zones de texte, boutons, cases à cocher), chacune ayant un nom. Lorsqu'un formulaire est "soumis"au serveur Web (une requête est envoyée au serveur), la requête comporte, entre autres, la liste des zones de saisie et leurs valeurs respectives.
Il y a deux "méthodes" pour soumettre un formulaire (4D accepte indifféremment l'une ou l'autre) :
• POST, généralement utilisée pour l'insertion de données dans le serveur Web — vers une base de données,
• GET, généralement utilisée pour l'interrogation du serveur Web — données en provenance d'une base de données.

Exemple

Un formulaire contient deux champs, vNOM et vVILLE, qui reçoivent les valeurs "MARTIN" et "PARIS". L'action associée au formulaire est "/4DACTION/WEBFORM".
• Si la méthode du formulaire est POST (cas le plus souvent utilisé), les données saisies ne seront pas visibles dans l'URL (c'est-à-dire http://127.0.0.1/4DACTION/WEBFORM).

• Si la méthode du formulaire est GET, les données seront visibles dans l'URL (c'est-à-dire http://127.0.0.1/4DACTION/WEBFORM?vNOM=MARTIN&vVILLE=PARIS).

La méthode WEBFORM peut être de la forme suivante :

```
TABLEAU TEXTE($tnoms;0)
TABLEAU TEXTE($tvaleurs;0)
LIRE VARIABLES FORMULAIRE WEB($tnoms;$tvaleurs)
```

On obtient alors :

```
$tnoms{1} = "vNOM"
$tnoms{2} = "vVILLE"
$tvaleurs{1} = "MARTIN"
$tvaleurs{2} = "PARIS"
```

Référence

Associer des objets 4D à des objets HTML, URLs et actions de formulaires.

STATISTIQUES DU CACHE WEB

STATISTIQUES DU CACHE WEB (pages; hits; usage)

Paramètre	Type		Description
pages	Tab Texte	←	Noms des pages les plus consultées
hits	Tab Entier long	←	Nombre de hits pour chaque page
usage	Numérique	←	Pourcentages du cache utilisé

Description

La commande STATISTIQUES DU CACHE WEB vous permet d'obtenir des informations sur les pages les plus consultées, chargées dans le cache du serveur Web. Par conséquent, ces statistiques concernent uniquement les pages statiques, les images GIF, les images JPEG <100 ko et les feuilles de style (.css).

Note : Pour plus d'informations sur le paramétrage du cache du serveur Web 4D, reportez-vous à la section Paramétrages du Serveur Web.

La commande remplit le tableau texte pages avec les noms des pages les plus consultées. Le tableau entier long hits reçoit le nombre de "hits" pour chaque page. La variable numérique usage reçoit le pourcentage du cache Web utilisé par chaque page.

Exemple

Vous souhaitez générer une page semi-dynamique affichant les statistiques d'utilisation du cache Web. Pour cela, dans une page HTML statique appelée "stats.shtm", vous placez la balise <!—4DACTION/STATS—>. Vous placez ensuite des références aux variables *vPages* et *vUsage*.
Dans la méthode projet *STATS*, écrivez le code suivant :

```
C_TEXTE($1)
TABLEAU TEXTE (pages;0)
TABLEAU ENTIER LONG (hits;0)
C_ENTIER LONG (vUsage)
```

```
STATISTIQUES DU CACHE WEB(pages;hits;vUsage)
vPages:=Caractere(1)
Boucle ($i;1;Taille tableau(pages))
        `Pour chaque page présente dans le cache
    vPages:=vPages+pages{$i}+"   "+Chaine(hits{$i})+"<br>"
        `Insertion du nom de la page et du code HTML
Fin de boucle
ENVOYER FICHIER HTML("stats.shtm")
        `Le contenu des pages suffixées ".shtm" est automatiquement analysé
```

Référence

Informations sur le site Web, Paramétrages du serveur Web.

TRAITER BALISES HTML

TRAITER BALISES HTML (données Entrée; données Sortie)

Paramètre	Type	Description
données Entrée	Texte | BLOB→	Données contenant des balises HTML à traiter
données Sortie	Texte | BLOB←	Données traitées

Description

La commande TRAITER BALISES HTML provoque le traitement par 4D des balises HTML 4D contenues dans le paramètre données Entrée (champ ou variable de type Texte ou BLOB) et retourne les données résultantes dans données Sortie.

Cette commande permet d'effectuer un traitement sur du code HTML balisé sans qu'il soit nécessaire que le serveur Web envoie une page HTML via une commande du type ENVOYER BLOB HTML ou qu'une page suffixée ".shtml" soit demandée via un URL. Il n'est même pas nécessaire que le serveur Web de 4D soit démarré.

Passez les données contenant les balises à traiter dans le paramètre données Entrée. Ce paramètre peut être un champ ou une variable de type Texte ou BLOB. A noter que si votre base fonctionne en mode compatibilité ASCII, le type BLOB est préférable car il n'est pas limité en nombre de caractères (le type Texte est en revanche limité à 32000 caractères). Si votre base fonctionne en mode standard (Unicode), le type Texte sera généralement suffisant (les paramètres peuvent recevoir jusqu'à 2 Go de texte).

Toutes les balises HTML de 4D sont prises en charge (4DVAR, 4DSCRIPT, 4DLOOP, etc.), quel que soit le mode de fonctionnement du serveur Web (contextuel ou sans contexte) — et même s'il n'est pas lancé.

Note : En cas d'utilisation de la balise 4DINCLUDE hors du cadre du serveur Web (process Web) :
• avec 4D en mode local et 4D Server, le dossier par défaut est le dossier contenant le fichier de structure de la base,
• avec 4D en mode distant, le dossier par défaut est le dossier contenant l'application 4D.

Après l'exécution de la commande, le paramètre données Sortie reçoit les données du paramètre données Entrée ainsi que le résultat du traitement des balises HTML 4D qu'il contenait, le cas échéant. Si données Entrée ne contenait pas de balises HTML 4D, le contenu de données Sortie est identique à celui de données Entrée.
Le paramètre données Sortie peut être un champ ou une variable, il doit simplement être du même type que le paramètre données Entrée.

Cette commande rend possible le stockage dans la base de valeurs issues d'un traitement de balises HTML avant qu'elles ne soient envoyées.
Elle permet également d'analyser des balises HTML 4D en-dehors de l'utilisation du serveur Web. En particulier, vous pouvez l'employer pour envoyer via 4D Internet Commands des courriels au format HTML contenant des traitements et/ou des références à des données contenues dans la base.

Exemple

Cet exemple illustre le fonctionnement de la commande :

```
C_BLOB($in)
C_BLOB($out)
C_TEXTE($in_text)
C_TEXTE(Var)
C_TEXTE(VarHTML)

Var:="<B>"
$in_text:="<p><!--#4DVAR Var→</p>"
TEXTE VERS BLOB($in_text;$in;UTF8 Texte sans longueur )
TRAITER BALISES HTML($in;$out)
VarHTML:=BLOB vers texte($out;UTF8 Texte sans longueur )
    ` VarHTML contient <p>&lt;B&gt;</p>
```

Référence

Balises HTML 4D.

Valider mot de passe digest Web

Valider mot de passe digest Web (nomUtilisateur; motDePasse) → Booléen

Paramètre	Type		Description
nomUtilisateur	Texte	→	Nom de l'utilisateur
motDePasse	Texte	→	Mot de passe de l'utilisateur
Résultat	Booléen	←	Vrai=Authentification correcte, Faux=Echec de l'authentification

Description

La commande Valider mot de passe digest Web permet de vérifier la validité des identifiants (nom et mot de passe) fournis par un utilisateur se connectant au serveur Web. Cette commande doit être utilisée dans la Méthode base Sur Authentification Web dans le cadre d'une authentification Web en mode Digest (cf. section Sécurité des connexions).

Passez dans les paramètres nomUtilisateur et motDePasse les identifiants de l'utilisateur conservés en local. La commande utilise ces identifiants pour générer une valeur qu'elle compare aux informations envoyées par le navigateur Web.

Si les valeurs sont identiques, la commande retourne Vrai. Sinon, elle retourne Faux.

Ce mécanisme vous permet de gérer et de maintenir par programmation votre propre système sécurisé d'accès au serveur Web. A noter que la validation Digest ne peut pas être utilisée conjointement avec les mots de passe 4D.

Note : Si le navigateur ne prend pas en charge l'authentification Digest, une erreur est retournée (erreur d'authentification).

Exemple

Exemple de méthode base Sur authentification Web en mode Digest

```
    ` Méthode base Sur authentification Web
C_TEXTE($1;$2;$5;$6;$3;$4)
C_TEXTE($utilisateur)
C_BOOLEEN($0)
$0:=Faux
$utilisateur:=$5
    `Pour des raisons de sécurité, refuser les noms qui contiennent @
Si (AvecJoker($utilisateur))
$0:=Faux
    `La méthode AvecJoker est décrite dans la section "Méthode base Sur
    `authentification Web"
Sinon
    CHERCHER([WebUsers];[WebUsers]User=$utilisateur)
    Si (OK=1)
        $0:=Valider mot de passe digest Web($utilisateur;[WebUsers]Mdp)
    Sinon
        $0:=Faux `Utilisateur inexistant
    Fin de si
Fin de si
```

Référence

Méthode base Sur authentification Web.

55

Sous-enregistrements

ALLER A DERNIER SOUS ENREGISTREMENT

version 11 (Modifiée)

ALLER A DERNIER SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dans laquelle se placer sur le dernier sous-enregistrement

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

ALLER A DERNIER SOUS ENREGISTREMENT sélectionne le dernier sous enregistrement de la sous-sélection courante de sousTable et en fait le sous-enregistrement courant. Si la sous-sélection courante est vide, ALLER A DERNIER SOUS ENREGISTREMENT ne fait rien.

Exemple

L'exemple suivant concatène le prénom et le nom de famille de chaque enfant dans la sous-table. Les noms sont copiés dans un tableau, Noms. Cet exemple ressemble à celui de la commande DEBUT SOUS ENREGISTREMENT mais celui-ci va du dernier sous-enregistrement au premier :

```
    ` Créer un tableau pour contenir les noms
TABLEAU TEXTE (taNoms; Sous enregistrements trouves ([Personnes]Enfants))
ALLER A DERNIER SOUS ENREGISTREMENT ([Personnes]Enfants)
    ` Commencer au dernier sous enregistrement et boucler pour chaque enfant
Boucle ($EL; 1; Sous enregistrements trouves ([Personnes]Enfants))
    Noms{$EL} := [Personnes]Enfants'Prénom + " " + [Personnes]Enfants'Nom
    SOUS ENREGISTREMENT PRECEDENT ([Personnes]Enfants)
Fin de boucle
```

Référence

DEBUT SOUS ENREGISTREMENT, SOUS ENREGISTREMENT PRECEDENT, SOUS ENREGISTREMENT SUIVANT.

APPLIQUER A SOUS SELECTION

APPLIQUER A SOUS SELECTION (sousTable; formule)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table à laquelle appliquer la formule
formule	Formule	→	Ligne de code ou méthode

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

APPLIQUER A SOUS SELECTION applique formule à chaque sous-enregistrement de la sous-sélection courante de sousTable. formule peut être une ligne d'instructions ou une méthode. Si formule entraîne la modification d'un sous-enregistrement, le sous-enregistrement modifié n'est sauvegardé que lorsque l'enregistrement parent est sauvegardé. Si la sous-sélection courante est vide, APPLIQUER A SOUS SELECTION ne fait rien.

APPLIQUER A SOUS SELECTION peut être utilisé pour récupérer et traiter des informations d'une sous-sélection d'enregistrements ou pour modifier une sous-sélection.

Exemple

L'exemple suivant met une lettre capitale aux prénoms du champ [Personnes]Enfants.

```
TOUT SELECTIONNER ([Personnes]Enfants)
APPLIQUER A SOUS SELECTION([Personnes]Enfants;[Personnes]Enfants'Nom:=
                    Majusc(Sous chaine([Personnes]Enfants'Nom;1;1))
                    +Minusc(Sous chaine([Personnes]Enfants'Nom;2)))
```

Note : La formule doit être écrite sur une seule ligne dans l'éditeur de méthodes de 4D.

Référence

CHERCHER SOUS ENREGISTREMENTS, EDITER FORMULE, STOCKER ENREGISTREMENT, TOUS LES SOUS ENREGISTREMENTS.

Avant sous enregistrement

Avant sous enregistrement (sousTable) → Booléen

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table pour laquelle vous testez si le pointeur se trouve avant la sous-sélection
Résultat	Booléen	←	Avant la sous-sélection (Vrai), sinon (Faux)

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

Avant sous enregistrement retourne Vrai lorsque le pointeur de sous-enregistrement courant se trouve avant le premier sous-enregistrement de sousTable. Avant sous enregistrement est utilisé pour vérifier si la commande SOUS ENREGISTREMENT PRECEDENT a déplacé le pointeur avant le premier sous-enregistrement. Si la sous-sélection courante est vide, Avant sous enregistrement retourne Vrai.

Exemple

L'exemple est une méthode objet pour un bouton. Lorsque l'utilisateur clique sur le bouton, le pointeur se place sur le sous-enregistrement précédent. Si le pointeur se trouve avant le premier sous-enregistrement, il est replacé sur le dernier sous-enregistrement :

```
SOUS ENREGISTREMENT PRECEDENT ([Personnes]Enfants)
    ` Aller au sous-enregistrement précédent
Si (Avant sous enregistrement ([Personnes]Enfants)   ` Si on y est déjà
    ALLER A DERNIER SOUS ENREGISTREMENT ([Personnes]Enfants)
        ` Aller au dernier sous-enregistrement
Fin de si
```

Référence

SOUS ENREGISTREMENT PRECEDENT.

CHERCHER SOUS ENREGISTREMENTS

CHERCHER SOUS ENREGISTREMENTS (sousTable; formule)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dans laquelle effectuer une recherche
formule	Booléen	→	Formule de recherche

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

CHERCHER SOUS ENREGISTREMENTS effectue une recherche dans sousTable et crée une nouvelle sous-sélection courante. CHERCHER SOUS ENREGISTREMENTS est la seule commande qui permet d'effectuer une recherche parmi des sous-enregistrements et qui retourne une sélection de sous-enregistrements. formule est appliquée à chaque sous-enregistrement de sousTable. Lorsque la formule est Vraie, le sous-enregistrement est ajouté à la nouvelle sous-sélection. Une fois que l'exécution de la formule est terminée, le premier sous-enregistrement de la sous-sélection devient le sous-enregistrement courant de sousTable.

N'oubliez pas que CHERCHER SOUS ENREGISTREMENTS effectue une recherche parmi les sous-enregistrements de la sous-table pour l'enregistrement parent courant uniquement, et non parmi tous les sous-enregistrements associés aux différents enregistrements de la table parente. CHERCHER SOUS ENREGISTREMENTS ne modifie pas l'enregistrement parent courant.

Typiquement, formule compare un sous-champ à une variable ou une constante, à l'aide d'un opérateur relationnel. formule peut comprendre plusieurs tests reliés par des opérateurs de type ET (&) ou de type OU (|). formule peut également être ou contenir une fonction. Le caractère Joker (@) peut être utilisé avec les arguments de type chaîne.

S'il n'y a pas d'enregistrement ni de sous-enregistrement courant, CHERCHER SOUS ENREGISTREMENTS ne fait rien.

Exemple

L'exemple suivant recherche les enfants âgés de plus de 10 ans :

```
CHERCHER SOUS ENREGISTREMENTS ([Personnes]Enfants; [Personnes]Enfants'Age>10)
```

Référence

Sous enregistrements trouves, TOUS LES SOUS ENREGISTREMENTS, TRIER SOUS ENREGISTREMENTS.

CREER SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dans laquelle vous voulez créer un sous-enregistrement

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

CREER SOUS ENREGISTREMENT crée un nouveau sous-enregistrement dans sousTable et en fait le sous-enregistrement courant. Ce nouveau sous-enregistrement n'est sauvegardé que lorsque l'enregistrement parent est lui-même sauvegardé. L'enregistrement parent peut être sauvegardé par une commande telle que STOCKER ENREGISTREMENT ou lorsque l'utilisateur le valide. S'il n'y a pas d'enregistrement courant, CREER SOUS ENREGISTREMENT ne fait rien. Pour ajouter un nouveau sous-enregistrement dans un formulaire de saisie de sous-enregistrements, utilisez AJOUTER SOUS ENREGISTREMENT.

Exemples

L'exemple suivant est la méthode objet d'un bouton. Lorsqu'elle est exécutée (lorsque l'utilisateur clique sur le bouton), elle crée de nouveaux sous-enregistrements pour des enfants. La boucle Repeter permet à l'utilisateur d'ajouter plusieurs enfants, jusqu'à ce qu'il clique sur Annuler. Le formulaire fait apparaître les enfants dans un sous-formulaire, mais ne permet pas d'y saisir directement des données car l'option "Saisissable" a été désactivée :

```
Repeter
        ` Répéter jusqu'à ce que l'utilisateur clique sur Annuler
    vEnfant := Demander("Prénom (annuler si terminé) :")
    Si (OK = 1)
            ` Création d'un nouveau sous-enregistrement pour un enfant
        CREER SOUS ENREGISTREMENT([Personnes]Enfants)
            ` Assignation du prénom de l'enfant au sous-champ
        [Personnes]Enfants'Prénom := vEnfant
    Fin de si
Jusque (OK = 0)
```

Référence

AJOUTER SOUS ENREGISTREMENT, MODIFIER SOUS ENREGISTREMENT, STOCKER ENREGISTREMENT, SUPPRIMER SOUS ENREGISTREMENT.

DEBUT SOUS ENREGISTREMENT

DEBUT SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table de laquelle charger le premier sous-enregistrement de la sélection courante

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

DEBUT SOUS ENREGISTREMENT charge le premier sous-enregistrement de la sélection courante de sousTable et en fait le sous-enregistrement courant. Toutes les commandes de recherche, de sélection et de tri font également du premier sous-enregistrement le sous-enregistrement courant. Si la sous-sélection courante est vide, DEBUT SOUS ENREGISTREMENT ne fait rien.

Exemple

L'exemple suivant met bout à bout les prénoms et les noms des enfants stockés dans une sous-table, puis les copie dans un tableau, nommé ttNoms :

```
` Création d'un tableau pour recevoir les noms
TABLEAU TEXTE (ttNoms; Sous enregistrements trouves ([Personnes]Enfants))
DEBUT SOUS ENREGISTREMENT ([Personnes]Enfants)
    ` Commençons au premier sous-enregistrement
    ` Effectuons une boucle par enfant
Boucle ($vlS; 1; Sous enregistrements trouves ([Personnes]Enfants))
    ttNoms{$vlS} := [Personnes]Enfants'Prénom+ " " + [Personnes]Enfants'Nom
    SOUS ENREGISTREMENT SUIVANT ([Personnes]Enfants)
Fin de boucle
```

Référence

ALLER A DERNIER SOUS ENREGISTREMENT, SOUS ENREGISTREMENT PRECEDENT, SOUS ENREGISTREMENT SUIVANT.

Fin sous enregistrement

Fin sous enregistrement (sousTable) → Booléen

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table pour laquelle tester si le pointeur de sous-enregistrement courant est placé au-delà du dernier sous-enregistrement sélectionné
Résultat	Booléen	←	Oui (Vrai) ou Non (Faux)

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

Fin sous enregistrement retourne VRAI lorsque le pointeur de sous-enregistrement courant se trouve après la fin de la sous-sélection courante de sousTable. Fin sous enregistrement a pour rôle de tester si l'utilisation de la commande SOUS ENREGISTREMENT SUIVANT place ou non le pointeur derrière le dernier sous-enregistrement de la sélection. Si la sous-sélection courante est vide, Fin sous enregistrement retourne Vrai.

Exemple

L'exemple suivant est la méthode objet d'un bouton. Lorsque l'utilisateur clique sur le bouton, le pointeur se place sur le sous-enregistrement suivant. Si le pointeur se retrouve derrière le dernier sous-enregistrement, il est replacé sur le premier sous-enregistrement :

```
SOUS ENREGISTREMENT SUIVANT ([Personnes]Enfants)
   ` Aller au sous-enregistrement suivant
Si (Fin sous enregistrement ([Personnes]Enfants)) ` Si nous sommes allés trop loin…
   DEBUT SOUS ENREGISTREMENT ([Personnes]Enfants)
      ` Aller au premier sous-enregistrement
Fin de si
```

Référence

SOUS ENREGISTREMENT SUIVANT.

SOUS ENREGISTREMENT PRECEDENT

SOUS ENREGISTREMENT PRECEDENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dans laquelle se placer sur le sous-enregistrement précédent de la sélection courante

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

SOUS ENREGISTREMENT PRECEDENT place le pointeur de sous-enregistrement courant sur le sous-enregistrement précédent dans la sous-sélection courante de sousTable. Si SOUS ENREGISTREMENT PRECEDENT place les pointeur avant le premier sous-enregistrement, Avant sous enregistrement retourne Vrai, et il n'y a plus de sous-enregistrement courant. Dans ce cas, utilisez DEBUT SOUS ENREGISTREMENT ou ALLER A DERNIER SOUS ENREGISTREMENT pour replacer le pointeur dans la sous-sélection courante. Si la sous-sélection courante est vide ou si Fin sous enregistrement retourne Vrai, SOUS ENREGISTREMENT PRECEDENT ne fait rien.

Exemple

Reportez-vous à l'exemple de la commande ALLER A DERNIER SOUS ENREGISTREMENT.

Référence

ALLER A DERNIER SOUS ENREGISTREMENT, DEBUT SOUS ENREGISTREMENT, SOUS ENREGISTREMENT SUIVANT.

SOUS ENREGISTREMENT SUIVANT

version 11 (Modifiée)

SOUS ENREGISTREMENT SUIVANT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dans laquelle se placer sur le sous-enregistrement suivant de la sélection courante

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

SOUS ENREGISTREMENT SUIVANT place le pointeur de sous-enregistrement courant sur le sous-enregistrement suivant de la sous-sélection courante de sousTable. Si SOUS ENREGISTREMENT SUIVANT place le pointeur de sous-enregistrement courant après le dernier sous-enregistrement, Fin sous enregistrement retourne VRAI, et il n'y a plus de sous-enregistrement courant. Lorsque Fin sous enregistrement retourne VRAI, utilisez DEBUT SOUS ENREGISTREMENT ou ALLER A DERNIER SOUS ENREGISTREMENT pour replacer le pointeur de sous-enregistrement courant dans la sous-sélection courante. Si la sous-sélection courante est vide, ou si Avant sous enregistrement retourne VRAI, SOUS ENREGISTREMENT SUIVANT ne fait rien.

Exemple

Reportez-vous à l'exemple de la commande DEBUT SOUS ENREGISTREMENT.

Référence

ALLER A DERNIER SOUS ENREGISTREMENT, DEBUT SOUS ENREGISTREMENT, SOUS ENREGISTREMENT PRECEDENT.

Sous enregistrements trouves

Sous enregistrements trouves (sousTable) → Numérique

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dont vous voulez connaître le nombre de sous-enregistrements
Résultat	Numérique	←	Nombre de sous-enregistrements de la sous-sélection courante

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

Sous enregistrements trouves retourne le nombre de sous-enregistrements de la sous-sélection courante de sousTable. Cette fonction s'applique uniquement aux sous-enregistrements de l'enregistrement courant. Elle est en quelque sorte l'équivalent de la fonction Enregistrements trouves pour les sous-enregistrements. Le résultat retourné est indéfini s'il n'existe pas d'enregistrement parent.

Exemple

L'exemple suivant sélectionne tous les sous-enregistrements puis affiche le nombre d'enfants pour l'enregistrement parent :

```
` Sélection de tous les enfants, puis affichage de leur nombre
TOUS LES SOUS ENREGISTREMENTS ([Personnes]Enfants)
ALERTE ("Le nombre d'enfants est : "+Chaine(Sous enregistrements trouves
                                                ([Personnes]Enfants)))
```

Référence

CHERCHER SOUS ENREGISTREMENTS, TOUS LES SOUS ENREGISTREMENTS.

SUPPRIMER SOUS ENREGISTREMENT

SUPPRIMER SOUS ENREGISTREMENT (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table de laquelle supprimer le sous-enregistrement courant

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

SUPPRIMER SOUS ENREGISTREMENT supprime le sous-enregistrement courant de sousTable. S'il n'y a pas de sous-enregistrement courant, SUPPRIMER SOUS ENREGISTREMENT ne fait rien. Après la suppression, la sous-sélection courante de sousTable est vide. En conséquence, la commande SUPPRIMER SOUS ENREGISTREMENT ne peut pas être utilisée pour parcourir une sous-sélection et supprimer des sous-enregistrements sélectionnés.

La suppression d'un sous-enregistrement n'est pas définitive tant que l'enregistrement parent n'est pas validé. La suppression d'un enregistrement parent entraîne automatiquement la suppression de tous ses sous-enregistrements.

Pour supprimer une sous-sélection, vous devez d'abord créer la sous-sélection, puis supprimer le premier sous-enregistrement, recréer la sous-sélection puis supprimer de nouveau le premier enregistrement, etc.

Exemples

(1) L'exemple suivant supprime tous les sous-enregistrements d'une sous-table :

```
TOUS LES SOUS ENREGISTREMENTS([Personnes]Enfants)
Tant que (Sous enregistrements trouves([Personnes]Enfants)>0)
    SUPPRIMER SOUS ENREGISTREMENT([Personnes]Enfants)
    TOUS LES SOUS ENREGISTREMENTS([Personnes]Enfants)
Fin tant que
```

(2) L'exemple suivant supprime de la sous-table [Personnes]Enfants tous les sous-enregistrements dans lesquels l'âge des enfants est supérieur ou égal à 12 ans :

```
TOUT SELECTIONNER([Personnes]) ` Sélection de tous les enregistrements
Boucle ($vELEnrg;1;Enregistrements trouves([Personnes])) ` Pour chaque enregistrement
        ` Recherche de tous les enregistrements ayant des sous-enregistrements
        `correspondant aux critères
    CHERCHER SOUS ENREGISTREMENTS([Personnes]Enfants;
                                  [Personnes]Enfants'Age>=12)
        ` Boucle jusqu'à ce que la recherche ne trouve plus d'enregistrement
    Tant que (Sous enregistrements trouves([Personnes]Enfants)>0)
            ` Supprimer le sous-enregistrement
        SUPPRIMER SOUS ENREGISTREMENT([Personnes]Enfants)
            ` On poursuit la recherche
        CHERCHER SOUS ENREGISTREMENTS([Personnes]Enfants;
                                      [Personnes]Enfants'Age>=12)
    Fin tant que
    STOCKER ENREGISTREMENT([Personnes]) ` Sauvegarde de l'enregistrement parent
    ENREGISTREMENT SUIVANT([Personnes])
Fin de boucle
```

Référence

CHERCHER SOUS ENREGISTREMENTS, Sous enregistrements trouves, STOCKER ENREGISTREMENT, TOUS LES SOUS ENREGISTREMENTS.

TOUS LES SOUS ENREGISTREMENTS

TOUS LES SOUS ENREGISTREMENTS (sousTable)

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table dans laquelle sélectionner tous les sous-enregistrements

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

TOUS LES SOUS ENREGISTREMENTS fait de tous les sous-enregistrements de sousTable la sous-sélection courante. S'il n'existe aucun enregistrement parent courant, TOUS LES SOUS ENREGISTREMENTS ne fait rien. Lorsqu'un enregistrement parent est chargé depuis le disque, la sous-sélection courante contient tous les sous-enregistrements. Une sous-sélection peut ne plus contenir tous les sous-enregistrements après un appel aux commandes AJOUTER SOUS ENREGISTREMENT, CHERCHER SOUS ENREGISTREMENTS, ou SUPPRIMER SOUS ENREGISTREMENT.

Exemple

Dans l'exemple suivant, on sélectionne tous les sous-enregistrements afin d'être sûr qu'ils soient tous inclus dans le total :

```
TOUS LES SOUS ENREGISTREMENTS ([Stats]Ventes)
VentesTotales := Somme ([Stats]Ventes'Francs)
```

Référence

CHERCHER SOUS ENREGISTREMENTS, Sous enregistrements trouves.

TRIER SOUS ENREGISTREMENTS

version 11 (Modifiée)

TRIER SOUS ENREGISTREMENTS (sousTable; sousChamp{; direction}{; sousChamp2; direction2; ...; sousChampN; directionN})

Paramètre	Type		Description
sousTable	Sous-table	→	Sous-table contenant le(s) sous-champ(s) à trier
sousChamp	Sous-champ	→	Sous-champ sur lequel effectuer le tri
direction	> ou <	→	> tri croissant ou < tri décroissant

Note de compatibilité : Les sous-tables ne sont plus prises en charge à compter de la version 11 de 4D. Un mécanisme de compatibilité assure le fonctionnement de cette commande dans les bases de données converties, toutefois il est fortement conseillé de remplacer les sous-tables par des tables liées standard.

Description

TRIER SOUS ENREGISTREMENTS trie la sous-sélection courante de sousTable. Seule la sous-sélection de sousTable contenue dans l'enregistrement parent courant est triée.

Le paramètre direction définit le sens du tri de sousChamp : ascendant ou descendant. Si direction a pour valeur le symbole “supérieur à” (>), les sous-enregistrements sont triés dans l'ordre ascendant. Si direction a pour valeur le symbole “inférieur à” (<), les sous-enregistrements sont triés dans l'ordre descendant. direction peut être omis (dans le cadre d'un tri sur un seul niveau) ; le tri est alors ascendant.

Vous pouvez spécifier plusieurs niveaux de tris en incluant autant de sous-champs et de symboles de tris que vous voulez.
Une fois le tri terminé, le premier sous-enregistrement de la sous-sélection triée devient le sous-enregistrement courant. Le tri des sous-enregistrements est une opération dynamique. Les sous-enregistrements ne sont jamais sauvegardés dans l'ordre où ils se trouvent après un tri.
S'il n'y a pas d'enregistrement courant ni de sous-enregistrement, TRIER SOUS ENREGISTREMENTS ne fait rien.
Si un formulaire contient un sous-formulaire dont l'impression est "limitée par le cadre", la commande n'a besoin d'être appelée qu'une seule fois avant l'impression, pendant l'événement formulaire Sur chargement de la méthode formulaire parente.

Exemple

L'exemple suivant trie la sous-table [Stats]Ventes dans un ordre croissant, sur la base du sous-champ VentesDollars :

```
TRIER SOUS ENREGISTREMENTS ([Stats]Ventes; [Stats]VentesDollars; >)
```

Référence

CHERCHER SOUS ENREGISTREMENTS.

56

SQL

version 11.3 (Modifiée)

4D, à compter de la version 11, comporte un moteur SQL intégré. Le programme inclut également un serveur SQL pouvant répondre aux requêtes d'autres applications 4D ou d'applications tierces (via le pilote ODBC de 4D).
Les différents modes d'accès au moteur SQL de 4D, le paramétrage du serveur SQL ainsi que les commandes et mots-clés utilisables dans les requêtes SQL sont décrits dans un manuel séparé, *Guide de référence 4D SQL*.

Le thème "SQL" regroupe les diverses commandes 4D relatives à l'exploitation du SQL dans 4D :
• contrôle du serveur SQL : LANCER SERVEUR SQL et ARRETER SERVEUR SQL
• accès direct au moteur SQL intégré : FIXER VALEUR CHAMP NULL, Valeur champ Null, CHERCHER PAR SQL.
• gestion des connexions aux sources de données internes ou externes (*SQL Pass-through*) : LISTE SOURCES DONNEES, Lire source donnees courante, SQL LOGIN, SQL LOGOUT.
• commandes de haut niveau pour la manipulation des données dans le cadre de connexions SQL directes ou via ODBC : Debut SQL, Fin SQL, SQL ANNULER CHARGEMENT, SQL CHARGER ENREGISTREMENT, SQL EXECUTER, SQL EXPORTER, SQL Fin de selection, SQL FIXER OPTION, SQL FIXER PARAMETRE, SQL IMPORTER, SQL LIRE DERNIERE ERREUR, SQL LIRE OPTION.

Principes de fonctionnement des commandes SQL de haut niveau
Les commandes SQL intégrées de 4D débutent par le préfixe "SQL". Elles appliquent les principes suivants :
• Vous pouvez utiliser ces commandes avec le moteur SQL interne de 4D ou dans une connexion externe ouverte directement ou via ODBC. La commande SQL LOGIN vous permet de définir le type de connexion à ouvrir.
• La portée d'une connexion est le process. Si vous souhaitez gérer plusieurs connexions simultanément, vous devez démarrer un process par SQL LOGIN.
La commande SQL ANNULER CHARGEMENT permet d'exécuter plusieurs requêtes SELECT dans la même connexion.
• Vous pouvez intercepter les erreurs ODBC éventuellement générées lors de l'exécution d'une des commandes SQL de haut niveau à l'aide la commande APPELER SUR ERREUR. La commande SQL LIRE DERNIERE ERREUR permet dans ce cas d'obtenir des informations supplémentaires.

Prise en charge du standard ODBC
Le standard ODBC (*Open DataBase Connectivity*) définit une librairie de fonctions standardisées. Ces fonctions permettent à une application telle que 4D d'accéder via le langage SQL à tout système de gestion de données compatible ODBC (bases de données, tableurs, autre application 4D, etc.).

Note : 4D permet également d'importer et d'exporter des données dans une source ODBC "manuellement" en mode Développement. Pour plus d'informations, reportez-vous au manuel *Mode Développement* de 4D.

Note : Les commandes SQL de haut niveau de 4D permettent de mettre en place des solutions simples pour faire communiquer les applications 4D et des sources de données ODBC. Si vos applications nécessitent une prise en charge plus étendue du standard ODBC, vous devrez acquérir le plug-in ODBC "bas niveau" de 4D, **4D ODBC Pro**.

Correspondance des types de données
Le tableau suivant liste les correspondances établies automatiquement par 4D entre les types de données 4D et SQL :

Type 4D	Type SQL
C_ALPHA	SQL_C_CHAR
C_TEXTE	SQL_C_CHAR
C_REEL	SQL_C_DOUBLE
C_DATE	SQL_C_TYPE_DATE
C_HEURE	SQL_C_TYPE_TIME
C_BOOLEEN	SQL_C_BIT
C_ENTIER	SQL_C_SHORT
C_ENTIER LONG	SQL_C_SLONG
C_BLOB	SQL_C_BINARY
C_IMAGE	SQL_C_BINARY
C_GRAPHE	SQL_C_BINARY

Référencer des expressions 4D dans les requêtes SQL
4D propose deux modes d'insertion des expressions 4D (variable, tableau, champ, pointeur, expression valide) dans les requêtes SQL : l'association directe et la définition de paramètres via SQL FIXER PARAMETRE.

L'association directe peut être effectuée de deux manières :
• insérer le nom de l'objet 4D entre les caractères << et >> dans le texte de la requête.
• faire précéder la référence de deux-point ":"
Exemples :

```
SQL EXECUTER("INSERT INTO emp (empno,ename)
                                    VALUES (<<vEmpno>>,<<vEname>>)")
SQL EXECUTER("SELECT age FROM People WHERE name= :vNom")
```

Dans ces exemples, les valeurs courantes des variables 4D *vEmpno*, *vEname* et *vNom* seront substituées aux paramètres lors de l'exécution de la requête. Cette solution fonctionne également avec les champs et les tableaux 4D.

Cette syntaxe, simple d'utilisation, présente toutefois l'inconvénient de n'être pas conforme à la norme SQL et de ne pas permettre l'utilisation de paramètres de sortie. Pour y remédier, vous pouvez utiliser la commande SQL FIXER PARAMETRE. Cette commande permet de définir chaque objet 4D à intégrer dans une requête ainsi que son mode d'utilisation (entrée, sortie ou les deux). La syntaxe produite est alors standard. Pour plus d'informations, reportez-vous à la description de la commande SQL FIXER PARAMETRE.

(1) Cet exemple exécute une requête SQL utilisant directement des tableaux 4D associés :

```
TABLEAU TEXTE(MonTabTexte;10)
TABLEAU ENTIER LONG(MonTabLong;10)

Boucle (vCounter;1;Taille tableau(MonTabTexte))
    MonTabTexte{vCounter}:="Texte"+Chaine(vCounter)
    MonTabLong{vCounter}:=vCounter
Fin de boucle
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field)
                                        VALUES (<<MonTabTexte>>, <<MonTabLong>>)"
SQL EXECUTER(SQLStmt)
```

(2) Cet exemple permet d'exécuter une requête SQL utilisant directement des champs 4D associés :

```
TOUT SELECTIONNER([Table 2])
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES
                                    (<<[Table 2]Champ1>"+">,<<[Table 2]Champ2>>)"
SQL EXECUTER(SQLStmt)
```

(3) Cet exemple permet d'exécuter une requête SQL passant directement une variable via un pointeur non dépointé :

```
C_ENTIER LONG($vLong)
C_POINTEUR($vPointeur)
$vLong:=1
$vPointeur:=->$vLong
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT Col1 FROM TEST WHERE Col1=:$vPointeur"
SQL EXECUTER(SQLStmt)
```

Récupération des valeurs dans 4D

La récupération dans le langage de 4D des valeurs issues de requêtes SQL peut être effectuée de deux manières :

• en utilisant les paramètres supplémentaires de la commande SQL EXECUTER (solution conseillée).

• en utilisant la clause INTO dans la requête SQL elle-même (solution à réserver aux cas particuliers).

Méthode base Sur authentification SQL

version 11.3 (Modifiée)

La Méthode base Sur authentification SQL permet de filtrer les requêtes adressées au serveur SQL intégré de 4D. Le filtrage peut être effectué sur la base du nom, du mot de passe ainsi que (facultativement) de l'adresse IP de l'utilisateur. Le développeur peut utiliser sa propre table d'utilisateurs ou celle des utilisateurs 4D pour évaluer les identifiants de connexion. Une fois la connexion authentifiée, la commande CHANGER UTILISATEUR COURANT doit être appelée afin de contrôler les accès de la requête au sein de la base 4D.

Lorsqu'elle existe, la Méthode base Sur authentification SQL est automatiquement appelée par 4D ou 4D Server à chaque connexion externe au serveur SQL. Le système interne de gestion des utilisateurs de 4D n'est alors pas sollicité. La connexion n'est acceptée que si la méthode base retourne Vrai dans $0 et si la commande CHANGER UTILISATEUR COURANT a été exécutée avec succès. Si l'une des deux conditions n'est pas remplie, la requête est rejetée.

Note : L'instruction SQL LOGIN(SQL_INTERNAL;$utilisateur;$motdepasse) ne déclenche pas l'appel de la Méthode base Sur authentification SQL car il s'agit dans ce cas d'une connexion interne.

La méthode base reçoit jusqu'à trois paramètres de type Texte, passés par 4D ($1, $2 et $3), et retourne un booléen, $0. Voici la description de ces paramètres :

Paramètres	Type	Description
$1	Texte	Nom d'utilisateur
$2	Texte	Mot de passe
$3	Texte	(Facultatif) Adresse IP du client à l'origine de la requête
$0	Booléen	Vrai = requête acceptée, Faux = requête rejetée

Vous devez déclarer ces paramètres de la manière suivante :

```
    ` Méthode base Sur authentification Web
C_TEXTE($1;$2;$3)
C_BOOLEEN($0)
... ` Code pour la méthode
```

Le mot de passe ($2) est reçu en texte standard.

Vous devez contrôler les identifiants de la connexion SQL dans la Méthode base Sur authentification SQL. Par exemple, vous pouvez contrôler le nom et le mot de passe à l'aide d'une table d'utilisateurs personnalisée. Si les identifiants sont valides, passez Vrai dans $0. Sinon, passez Faux dans $0, dans ce cas la connexion est rejetée.

Par défaut, $0 vaut Faux. Si la Méthode base Sur authentification SQL existe et si $0 n'est pas défini, toutes les connexions sont donc rejetées.

Si vous avez passé Vrai dans $0, vous devez ensuite appeler avec succès la commande CHANGER UTILISATEUR COURANT dans la Méthode base Sur authentification SQL pour que la requête soit acceptée et que 4D ouvre une session SQL pour l'utilisateur.
L'utilisation de la commande CHANGER UTILISATEUR COURANT permet de mettre en place un système d'authentification virtuelle ayant comme double avantage le contrôle des actions au sein de la connexion et le masquage pour l'extérieur des identifiants de la connexion dans la session SQL 4D.

Note : Si la Méthode base Sur authentification SQL n'existe pas, la connexion est évaluée à l'aide du système intégré de gestion des utilisateurs de 4D s'il est actif, c'est-à-dire si un mot de passe a été attribué au Super_Utilisateur. Si le système n'est pas actif, les utilisateurs sont connectés avec les droits du Super_Utilisateur (accès libre).

Cet exemple de Méthode base Sur authentification SQL vérifie que la demande de connexion provient du réseau interne, valide les identifiants puis affecte les droits d'utilisateur "sql_user" pour la session SQL.

```
C_TEXTE($1;$2;$3)
C_BOOLEEN ($0)
    `$1 : utilisateur
    `$2 : mot de passe
    `{$3 : Adresse IP du client}
APPELER SUR ERREUR ("SQL_error")
Si (checkInternalIP($3))
    `La méthode checkInternalIP vérifie que l'adresse IP est interne
    Si ($1="victor") & ($2="hugo")
        CHANGER UTILISATEUR COURANT("sql_user";"")
        Si (OK=1)
            $0:=Vrai
        Sinon
            $0:=Faux
        Fin de si
    Sinon
        $0:=Faux
    Fin de si
Sinon
    $0:=Faux
Fin de si
```

Référence

CHANGER UTILISATEUR COURANT.

ARRETER SERVEUR SQL

ARRETER SERVEUR SQL

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande ARRETER SERVEUR SQL stoppe le serveur SQL intégré de l'application 4D sur laquelle elle a été exécutée.

Si le serveur SQL était lancé, toutes les connexions SQL sont interrompues et le serveur n'accepte plus aucune requête SQL externe. Si le serveur SQL n'était pas lancé, la commande ne fait rien.

Note : Cette commande n'influe pas sur le fonctionnement du moteur SQL interne de 4D. Le moteur SQL toujours disponible pour les requêtes internes.

Référence

LANCER SERVEUR SQL.

CHERCHER PAR SQL

version 11

CHERCHER PAR SQL ({uneTable; }formuleSQL)

Paramètre	Type		Description
uneTable	Table	→	Table de laquelle retourner une sélection d'enregistrements ou Table par défaut si ce paramètre est omis
formuleSQL	Chaîne	→	Formule de recherche SQL valide représentant la clause WHERE de la requête SELECT

Description

La commande CHERCHER PAR SQL permet de tirer directement parti du moteur SQL intégré de 4D. Elle exécute une requête SELECT simple qui peut être écrite ainsi :

```
SELECT *
    FROM table
    WHERE <formuleSQL>
```

table est le nom de la table passé en premier paramètre et formuleSQL la chaîne de recherche passée en deuxième paramètre.

Par exemple, l'instruction suivante :

```
CHERCHER PAR SQL([Employees];"name='smith'")
```

équivaut à la requête SQL :

```
SELECT * FROM Employees WHERE "name='smith'"
```

La commande CHERCHER PAR SQL est semblable à la commande CHERCHER PAR FORMULE. Elle effectue une recherche parmi les enregistrements de la table définie. Elle modifie la sélection courante de table pour le process courant et fait du premier enregistrement de la nouvelle sélection le nouvel enregistrement courant.

Note : La commande CHERCHER PAR SQL ne peut pas être utilisée dans le contexte d'une connexion SQL externe, elle s'adresse directement au moteur SQL intégré de 4D.

CHERCHER PAR SQL applique formuleSQL à chaque enregistrement de la sélection de la table. formuleSQL est une expression booléenne qui doit retourner VRAI ou FAUX. Comme vous le savez peut-être, dans la norme SQL, une condition de recherche peut avoir un résultat VRAI, FAUX ou NULL. Tous les enregistrements (rows) pour lesquels la condition de recherche retourne VRAI sont inclus dans la nouvelle sélection courante.

L'expression formuleSQL peut être simple, comme par exemple la comparaison d'un champ (colonne) à une valeur ; elle peut également être complexe, comme la réalisation d'un calcul. Comme CHERCHER PAR FORMULE, CHERCHER PAR SQL peut évaluer des valeurs dans les tables liées (cf. exemple 4). formuleSQL doit être une instruction SQL valide, conforme à la norme SQL-2 et tenant compte de l'implémentation actuelle du SQL dans 4D. Pour plus d'information la prise en charge du SQL dans 4D, reportez-vous au manuel *Guide de référence 4D SQL*.

Le paramètre formuleSQL peut contenir des références à des expressions 4D. La syntaxe à utiliser est la même que pour les commandes SQL intégrées ou le code inclus dans les balises Debut SQL/Fin SQL, c'est-àdire : <<MaVar>> ou :MaVar
Pour plus d'informations sur ce point, reportez-vous à la section Commandes du thème Source de données.

Note : Cette commande est compatible avec les commandes FIXER LIMITE RECHERCHE et FIXER DESTINATION RECHERCHE.

A propos des liens
CHERCHER PAR SQL n'utilise pas les liens entre les tables définis dans l'éditeur de structure de 4D. Si vous souhaitez tirer parti des données liées, vous devez ajouter une clause JOIN dans la requête. Par exemple, considérons la structure suivante, dans laquelle un lien N vers 1 relie les champs [Personnes]Ville à [Villes]Nom :

[Personnes]
Nom
Ville
[Villes]
Nom
Population

Avec la commande CHERCHER PAR FORMULE, vous pourriez écrire :

CHERCHER PAR FORMULE([Personnes];[Villes]Population>1000)

Avec CHERCHER PAR SQL, vous devez écrire l'instruction suivante, que le lien existe ou non :

CHERCHER PAR SQL([Personnes];"personnes.ville=villes.nom AND
villes.population>1000")

Note : Les liens 1 vers N et N vers N sont également traités par CHERCHER PAR SQL d'une manière différente de CHERCHER PAR FORMULE.

Exemples

(1) Cet exemple recherche les bureaux dont les ventes sont supérieures à 100. La requête SQL est :

```
SELECT *
    FROM Bureaux
    WHERE Ventes > 100
```

En utilisant la commande CHERCHER PAR SQL :

```
C_ALPHA(30;$formuleRequete)
$formuleRequete:="Ventes > 100"
CHERCHER PAR SQL([Bureaux];$formuleRequete)
```

(2) Cet exemple recherche les commandes comprises entre 3000 et 4000. La requête SQL est :

```
SELECT *
    FROM Commandes
    WHERE Total BETWEEN 3000 AND 4000
```

En utilisant la commande CHERCHER PAR SQL :

```
C_ALPHA(40;$formuleRequete)
$formuleRequete:="Total BETWEEN 3000 AND 4000"
CHERCHER PAR SQL([Ventes];$formuleRequete)
```

(3) Cet exemple montre comment trier le résultat de la requête sur un critère spécifique. La requête SQL est :

```
SELECT *
    FROM Personnes
    WHERE Ville ='Paris'
        ORDER BY Nom
```

En utilisant la commande CHERCHER PAR SQL :

```
C_ALPHA(40;$formuleRequete)
$formuleRequete:="Ville = 'Paris' ORDER BY Nom"
CHERCHER PAR SQL([Personnes];$formuleRequete)
```

(4) Cet exemple montre une requête utilisant des tables liées dans 4D. Via le SQL vous devez utiliser un JOIN pour recréer cette relation. Considérons les deux tables suivantes :

[Factures] avec les champs (colonnes) suivants :
- ID_Fact : Entier long
- Date_Fact : Date
- Total : Réel

[Lignes_Factures] avec les champs (colonnes) suivants :
- ID_Ligne : Entier long
- ID_Fact : Entier long
- Code : Alpha (10)

Un lien de N vers 1 relie le champ [Lignes_Factures]ID_Fact au champ [Factures]ID_Fact. Avec la commande CHERCHER PAR FORMULE, vous pourriez écrire :

```
CHERCHER PAR FORMULE([Lignes_Factures];([Lignes_Factures]Code="FX-200") &
                                        (Mois de([Factures]Date_Fact)=4))
```

La requête SQL est :

```
SELECT ID_Ligne
    FROM Lignes_Factures, Factures
    WHERE Lignes_Factures.ID_Fact=Factures.ID_Fact
        AND Lignes_Factures.Code='FX-200'
        AND MONTH(Factures.Date_Fact) = 4
```

En utilisant la commande CHERCHER PAR SQL :

```
C_ALPHA(40;$formuleRequete)
$formuleRequete:="Lignes_Factures.ID_Fact=Factures.ID_Fact AND
                  Lignes_Factures.Code='FX-200' AND MONTH(Factures.Date_Fact)=4"
CHERCHER PAR SQL([Lignes_Factures];$formuleRequete)
```

Référence

CHERCHER PAR FORMULE.

Variables et ensembles système

Si le format de la condition de recherche est correct, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0, le résultat de la commande est une sélection vide et une erreur est retournée. Cette erreur peut être interceptée par une méthode installée à l'aide de la commande APPELER SUR ERREUR.

Debut SQL

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Debut SQL est un *mot-clé* permettant d'indiquer dans l'éditeur de méthodes le début d'une séquence de commandes SQL, qui devront être interprétées par la source de données courante du process (moteur SQL intégré de 4D ou toute source définie via la commande SQL LOGIN).

Une séquence de commandes SQL initiée par Debut SQL doit être refermée par le mot-clé Fin SQL.

Les principes de fonctionnement de ces mots-clés sont les suivants :
• Vous pouvez placer un ou plusieurs blocs de balises Debut SQL/Fin SQL dans la même méthode. Vous pouvez générer des méthodes entièrement composées de code SQL ou mixer du code 4D et du code SQL dans la même méthode.
• Vous pouvez écrire plusieurs instructions SQL sur une même ligne ou sur différentes lignes en les séparant par un “;”. Par exemple, vous pouvez écrire :

```
Debut SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (‘Henry’,40);
   INSERT INTO SALESREPS (NAME, AGE) VALUES (‘Bill’,35)
Fin SQL
```

ou :

```
Debut SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (‘Henry’,40);INSERT INTO
                                                SALESREPS (NAME, AGE) VALUES (‘Bill’,35)
Fin SQL
```

A noter que le débogueur de 4D évaluera le code SQL ligne par ligne. Dans certains cas, il peut être préférable d’utiliser plusieurs lignes.

Référence

Fin SQL, Lire source donnees courante, SQL LOGIN.

Fin SQL

Fin SQL

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

Fin SQL est un *mot-clé* indiquant dans l'éditeur de méthodes la fin d'une séquence de commandes SQL.

Une séquence d'instructions SQL doit être encadrée par les mot-clés Debut SQL et Fin SQL. Pour plus d'informations, reportez-vous à la description du mot-clé Debut SQL.

Référence

Debut SQL.

FIXER VALEUR CHAMP NULL

FIXER VALEUR CHAMP NULL (unChamp)

Paramètre	Type		Description
unChamp	Champ	→	Champ auquel attribuer la valeur NULL

Description

La commande FIXER VALEUR CHAMP NULL attribue la valeur NULL au champdésigné par le paramètre unChamp.

La valeur NULL est exploitée par le moteur SQL de 4D. Pour plus d'informations, reportez-vous au manuel *Guide de référence 4D SQL*.

Note : Il est possible d'interdire la valeur Null pour les champs 4D au niveau de l'éditeur de Structure (cf. manuel *Mode Développement*).

Référence

Valeur champ Null.

LANCER SERVEUR SQL

LANCER SERVEUR SQL

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande LANCER SERVEUR SQL démarre le serveur SQL intégré de l'application 4D sur laquelle elle a été exécutée. Une fois lancé, le serveur SQL peut répondre aux requêtes SQL externes.

Note : Cette commande n'influe pas sur le fonctionnement du moteur SQL interne de 4D. Le moteur SQL est toujours disponible pour les requêtes internes.

Référence

ARRETER SERVEUR SQL.

Variables et ensembles système

Si le serveur SQL a été correctement lancé, la variable système *OK* prend la valeur 1, sinon elle prend la valeur 0.

Lire source donnees courante

Lire source donnees courante → Chaîne

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Chaîne	←	Nom de la source de données en cours d'utilisation

Description

La commande Lire source donnees courante retourne le nom de la source de données courante de l'application. La source de données courante reçoit les requêtes SQL exécutées au sein de structures Debut SQL/Fin SQL.
Lorsque la source de données courante est la base 4D locale, la commande retourne la chaîne ";DB4D_SQL_LOCAL;", correspondant à la valeur de la constante SQL_INTERNAL (thème "SQL").
Cette commande vous permet de contrôler la source de données courante, généralement avant d'exécuter une requête SQL.

Référence

Debut SQL, Fin SQL, LISTE SOURCES DONNEES, SQL LOGIN, SQL LOGOUT.

LISTE SOURCES DONNEES

SQL

version 11

LISTE SOURCES DONNEES (typeSource; tabNomsSources; tabPilotes)

Paramètre	Type		Description
typeSource	Entier long	→	Type de source : utilisateur ou système
tabNomsSources	Tab Texte	←	Tableau des noms de sources de données
tabPilotes	Tab Texte	←	Tableau des pilotes des sources

Description

La commande LISTE SOURCES DONNEES retourne dans les tableaux tabNomsSources et tabPilotes les noms et les pilotes des sources de données de type typeSource définies dans le gestionnaire ODBC du système d'exploitation.

4D vous permet de vous connecter directement via le langage à une source de données ODBC externe et d'exécuter des requêtes SQL au sein d'une structure Debut SQL/Fin SQL. Le principe d'utilisation est le suivant : la commande LISTE SOURCES DONNEES permet d'obtenir la liste des sources de données présentes sur le poste. La commande SQL LOGIN permet alors de désigner la source à utiliser. Vous pouvez ensuite exécuter des requêtes SQL dans une structure Debut SQL/Fin SQL sur la source "courante". Pour effectuer à nouveau des requêtes sur le moteur interne de 4D, il suffit de passer la commande SQL LOGOUT. Pour plus d'informations sur les commandes SQL dans l'éditeur de méthodes, reportez-vous au manuel *Guide de référence 4D SQL*.

Passez dans typeSource le type de source de données que vous souhaitez obtenir. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème "SQL" :

Constante	Type	Valeur
Source de données utilisateur	Entier long	1
Source de données système	Entier long	2

Note : Cette commande ne prend pas en compte les sources de données de type fichier.

La commande remplit et dimensionne les tableaux tabNomsSources et tabPilotes avec les valeurs correspondantes.

Note : Si vous souhaitez vous connecter à une source de données 4D externe via ODBC, vous devez au préalable installer le pilote 4D ODBC sur votre poste. Pour plus d'informations, reportez-vous au manuel d'installation de 4D ODBC Driver.

Exemple

Cet exemple utilise une source de données utilisateur :

```
TABLEAU TEXTE(tdsn;0)
TABLEAU TEXTE(tdsnPilotes;0)
LISTE SOURCES DONNEES(Source de données utilisateur;tdsn;tdsnPilotes)
```

Référence

Lire source donnees courante, SQL LOGIN, SQL LOGOUT.

Variables et ensembles système

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est générée.

SQL ANNULER CHARGEMENT

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

La commande SQL ANNULER CHARGEMENT met fin à la requête SELECT courante et initialise les paramètres du curseur.

Cette commande permet d'exécuter plusieurs requêtes SELECT au sein d'une même connexion (c'est-à-dire un même curseur) initiée par la commande SQL LOGIN.

Exemple

Dans cet exemple, deux requêtes sont exécutées dans la même connexion :

```
C_BLOB(Monblob)
C_TEXTE(MonTexte)
SQL LOGIN("mysql";"root";"")

SQLStmt:="SELECT blob_field FROM app_testTable"
SQL EXECUTER(SQLStmt;Monblob)
Tant que(Non(SQL Fin de selection))
    SQL CHARGER ENREGISTREMENT
Fin tant que

    `Réinitialisation du curseur
SQL ANNULER CHARGEMENT
SQLStmt:="SELECT Name FROM Employee"
SQL EXECUTER(SQLStmt;MonTexte)
Tant que(Non(SQL Fin de selection))
    SQL CHARGER ENREGISTREMENT
Fin tant que
```

Référence

SQL CHARGER ENREGISTREMENT, SQL LOGIN.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL CHARGER ENREGISTREMENT

version 2004

SQL CHARGER ENREGISTREMENT {(nombreEnr)}

Paramètre	Type		Description
nombreEnr	Entier	→	Nombre d'enregistrements à charger

Description

La commande SQL CHARGER ENREGISTREMENT récupère dans 4D un ou plusieurs enregistrement(s) provenant de la source de données ouverte dans la connexion courante.

Le paramètre facultatif nombreEnr permet de définir le nombre d'enregistrements à récupérer :
• Si vous omettez ce paramètre, la commande récupèrera l'enregistrement courant dans la source de données. Ce principe correspond à la récupération des données dans une boucle où un enregistrement est reçu à la fois.
• Si vous passez une valeur entière dans nombreEnr, la commande récupérera nombreEnr enregistrements.
• Si vous passez la constante SQL Tous les enregistrements (ou la valeur -1), la commande récupèrera tous les enregistrements de la table.

Note : Ces deux derniers paramétrages n'ont de sens que si les données récupérées sont associées à des tableaux ou des champs 4D.

Référence

SQL ANNULER CHARGEMENT, SQL EXECUTER.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL EXECUTER

version 2004

SQL EXECUTER (instructionSQL{; objetLié}{; objetLié2; ...; objetLiéN})

Paramètre	Type		Description
instructionSQL	Texte	→	Commande SQL à exécuter
objetLié	Var | Champ	←	Réception du résultat (si nécessaire)

Description

La commande SQL EXECUTER permet d'exécuter une commande SQL et d'associer le résultat à des objets 4D (tableaux, variables ou champs) liés.

Pour que la commande puisse être exécutée, une connexion valide doit être définie dans le process courant.

Le paramètre instructionSQL contient la commande SQL à exécuter. Le paramètre objetLié reçoit les résultats. Les objets sont liés dans l'ordre de la colonne, ce qui signifie que les éventuelles colonnes distantes supplémentaires sont ignorées.

Si des champs 4D sont passés dans le(s) paramètre(s) objetLié, la commande créera des enregistrements et les sauvegardera automatiquement. Les champs doivent appartenir à la même table (il n'est pas possible de passer un champ de la table 1 et un champ de la table 2 dans le même appel). Si des champs de tables différentes sont passés, une erreur est générée.

Si vous passez des tableaux ou des variables 4D dans le(s) paramètre(s) objetLié, il est conseillé de les déclarer préalablement à l'appel de la commande afin de contrôler le type de données traitées. Les tableaux sont redimensionnés automatiquement si nécessaire.

Dans le cas d'une variable 4D, un seul enregistrement est récupéré à la fois.

Note : Pour plus d'informations sur le référencement des expressions 4D dans les requêtes SQL, reportez-vous à la section Présentation des commandes du thème SQL.

Exemples

(1) Dans cet exemple, nous récupérons la colonne *ename* de la table *emp* dans la source de données. Le résultat est stocké dans le champ 4D [Employés]Nom. Les enregistrements 4D seront créés automatiquement :

```
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTER(SQLStmt;[Employés]Nom)
SQL CHARGER ENREGISTREMENT(SQL Tous les enregistrements)
```

(2) Pour mieux contrôler la création des enregistrements, il est possible d'inclure le code au sein d'une transaction et de ne la valider que si le déroulement de l'opération s'est avéré satisfaisant :

```
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT alpha_field FROM app_testTable"
DEBUT TRANSACTION
SQL EXECUTER(SQLStmt;[Table 2]Champ1)
Tant que(Non(SQL Fin de selection))
    SQL CHARGER ENREGISTREMENT
    ... `Placer ici le code de validation des données
Fin tant que
VALIDER TRANSACTION  `Validation de la transaction
```

(3) Dans cet exemple, nous récupérons la colonne *ename* de la table *emp* dans la source de données. Le résultat est stocké dans le tableau tNoms. Nous récupérons les enregistrements 10 par 10.

```
TABLEAU ALPHA(30;tNoms;20)
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTER(SQLStmt;tNoms)
Tant que(Non(SQL Fin de selection))
    SQL CHARGER ENREGISTREMENT(10)
Fin tant que
```

(4) Dans cet exemple, nous récupérons les colonnes *ename* et *job* de la table *emp* pour un ID spécifique (clause WHERE) de la source de données. Le résultat est stocké dans les variables 4D vNom and vJob. Seul le premier enregistrement est récupéré.

```
SQLStmt:="SELECT ename, job FROM emp WHERE id = 3"
SQL EXECUTER(SQLStmt;vName;vJob)
SQL CHARGER ENREGISTREMENT
```

(5) Dans cet exemple, nous récupérons la colonne *Champ_Blob* de la table *Test* dans la source de données. Le résultat est stocké dans une variable BLOB dont la valeur est mise à jour à chaque chargement d'enregistrement.

```
C_BLOB(MonBlob)
SQL LOGIN
SQL EXECUTER("SELECT Champ_Blob FROM Test";MonBlob)
Tant que(Non(SQL Fin de selection))
        `On parcourt le résultat
    SQL CHARGER ENREGISTREMENT
        `La valeur de MonBlob est mise à jour à chaque appel
Fin tant que
```

Référence

SQL CHARGER ENREGISTREMENT.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL EXPORTER (tableSource{; projet{; *}})

Paramètre	Type		Description
tableSource	Chaîne	→	Nom de la table dans la source de données ODBC
projet	BLOB	→	Contenu du projet d'exportation
		←	Nouveau contenu du projet (si * est passé)
*	*	→	Affichage de la boîte de dialogue d'exportation et mise à jour du projet

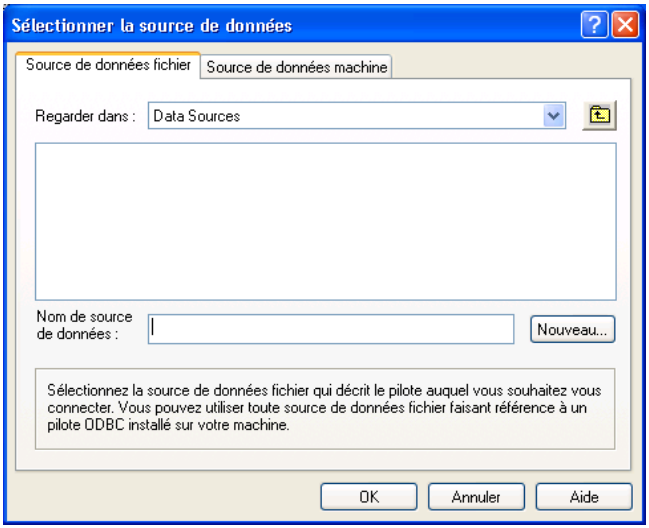
Description

La commande SQL EXPORTER permet d'exporter des données dans la table tableSource d'une source ODBC externe. Les paramètres de connexion (nom de la source, utilisateur et mot de passe) sont inclus dans le BLOB projet.

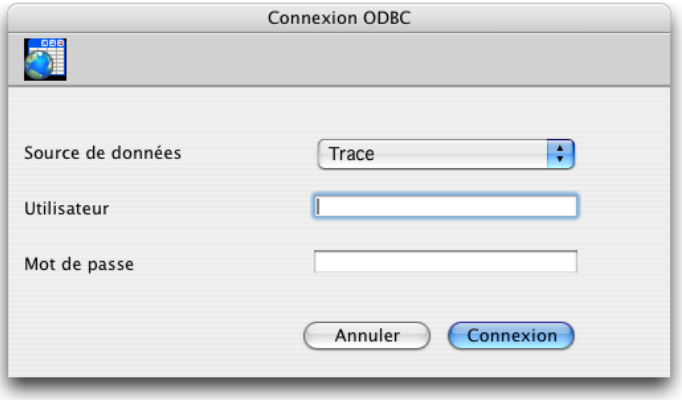
Notes :
• Le *projet* contient tous les paramètres de l'exportation, notamment la source de données et les tables et champs exportés. Vous définissez ces paramètres dans la boîte de dialogue d'exportation ODBC, puis vous pouvez éventuellement les sauvegarder dans un fichier sur disque. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.
• Les projets générés dans la boîte de dialogue d'exportation ODBC ne sont pas compatibles avec les commandes ou la boîte de dialogue d'exportation standard de 4D.
• Cette commande ne peut pas être utilisée dans le cadre d'une connexion avec le moteur SQL interne de 4D.

Si vous ne passez pas le paramètre facultatif projet, SQL EXPORTER provoque l'affichage de la boîte de dialogue de sélection de la source de données ODBC :

Windows



Mac OS



Une fois la source sélectionnée, la boîte de dialogue d'exportation ODBC de 4D apparaît, permettant à l'utilisateur de configurer l'opération. Si l'utilisateur clique sur le bouton **Annuler** dans l'une des deux boîtes de dialogue, l'exécution est stoppée et la variable système OK prend la valeur 0.

Si vous passez dans le paramètre projet un BLOB contenant un projet d'export ODBC valide, l'exportation s'effectue directement, sans intervention de l'utilisateur. Pour cela, il vous suffit de charger un projet préalablement sauvegardé sur disque dans le champ ou la variable BLOB que vous passez dans le paramètre projet, à l'aide de la commande DOCUMENT VERS BLOB.

Vous pouvez également utiliser la commande SQL EXPORTER avec un paramètre projet vide et le paramètre optionnel *, puis stocker le paramètre projet dans un champ BLOB (cf. ci-dessous). Cette solution permet, d'une part, de le conserver avec le fichier de données, et d'autre part d'éviter la phase de chargement dans un BLOB depuis le disque.

Note : Reportez-vous à la commande EXPORTER DONNEES pour un exemple de définition de projet vide.

Le paramètre optionnel *, s'il est spécifié, provoque l'affichage de la boîte de dialogue de paramétrage d'exportation ODBC avec les paramétrages éventuellement définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, dans ce cas le paramètre projet contient, après la fermeture de la boîte de dialogue, les paramètres du "nouveau" projet. Vous pouvez alors le stocker dans un champ BLOB, sur disque, etc.

Référence

SQL IMPORTER.

Variables et ensembles système

Si l'utilisateur clique sur le bouton **Annuler** dans une des boîtes de dialogue (de sélection de source de données ou de paramétrage de l'export), la variable système OK prend la valeur 0. Si l'exportation se déroule correctement, la variable système OK prend la valeur 1.

SQL Fin de selection → Booléen

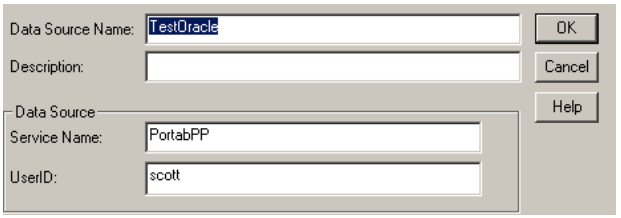
Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Limites de l'ensemble de recherche atteintes

Description

La commande SQL Fin de selection indique si les limites de l'ensemble résultat ont été atteintes.

Exemple

Le code ci-dessous se connecte à une source de données externe (Oracle) à l'aide des paramètres suivants :




```
C_TEXTE(vName)

SQL LOGIN("TestOracle";"scott";"tiger")
Si (OK=1)
    SQL EXECUTER("SELECT ename FROM emp";vName)
    Tant que(Non(SQL Fin de selection))
        SQL CHARGER ENREGISTREMENT
    Fin tant que
    SQL LOGOUT
Fin de si
```

Cet exemple retournera dans la variable 4D vName les noms (*ename*) stockés dans la table nommée *emp*.

SQL FIXER OPTION (option; valeur)

Paramètre	Type		Description
option	Entier long	→	Numéro d'option à définir
valeur	Entier long | Chaîne	→	Nouvelle valeur de l'option

Description

La commande SQL FIXER OPTION permet de modifier la valeur de l'option passée dans le paramètre option.

Vous pouvez passer dans option l'une des constantes suivantes, placées dans le thème "SQL" :

Constante	Description et valeurs possibles
SQL Asynchrone (1)	0 = connexion synchrone (valeur par défaut), 1 (ou valeur différente de 0) = connexion asynchrone
SQL Nombre maxi lignes (2)	Nombre maximum de lignes dans l'ensemble résultant (utilisé pour les prévisualisations)
SQL Longueur maxi données (3)	Longueur maximale des données retournées
SQL Timeout requête (4)	Durée maximale d'attente de la réponse lors de l'exécution de la commande SQL EXECUTER. Valeurs : durée en secondes Par défaut : pas de timeout
SQL Timeout connexion (5)	Durée maximale d'attente lors de l'exécution de la commande SQL LOGIN. Cette valeur doit être fixée avant l'ouverture de la connexion pour être prise en compte Valeurs possibles : durée en secondes Par défaut : pas de timeout
SQL Jeu de caractères (100)	Encodage du texte utilisé pour les requêtes envoyées aux sources externes (via le *SQL pass-through*) La modification est effective pour le process courant et la connexion courante Valeurs possibles : identifiant *MIBEnum* (cf. note 2) ou chaîne "WCHAR" (cf. note 3) Par défaut : 106 (UTF-8)

Notes :

1. Lorsque vous travaillez avec le moteur SQL interne de 4D, l'option SQL Asynchrone est inutile. En effet, ce type de connexion est toujours synchrone.
2. Les numéros *MIBEnum* sont référencés à l'adresse suivante : http://www.iana.org/assignments/character-sets.

3. Lorsque la chaîne "WCHAR" est passée comme valeur à SQL Jeu de caractères, l'encodage utilisé par le serveur SQL de 4D est automatiquement adapté à la plate-forme d'exécution :
- sous Windows, UTF-16 est utilisé,
- sous Mac OS, UTF-32 est utilisé.

Référence

SQL LIRE OPTION.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL FIXER PARAMETRE (objet; typeParam)

Paramètre	Type		Description
objet	Objet 4D	→	Objet 4D à utiliser (variable, tableau ou champ)
typeParam	Entier long	→	Type du paramètre

Description

La commande SQL FIXER PARAMETRE permet d'utiliser la valeur d'une variable, d'un tableau ou d'un champ 4D dans les requêtes SQL.

Note : Il est également possible d'insérer directement le nom d'un objet 4D à utiliser (variable, tableau ou champ) entre les caractères << et >> dans le texte de la requête (cf. exemple 1). Pour plus d'informations sur ce point, reportez-vous à la section Présentation des commandes du thème SQL.

• Passez dans le paramètre objet l'objet 4D (variable, tableau ou champ) à utiliser dans la requête.
• Passez dans le paramètre typeParam le type SQL du paramètre. Vous pouvez passer une valeur ou utiliser l'une des constantes suivantes, placées dans le thème "SQL" :

Constante	Type	Valeur
SQL Paramètre entrée	Entier long	1
SQL Paramètre entrée sortie	Entier long	2
SQL Paramètre sortie	Entier long	4

La valeur de l'objet 4D est substituée au caractère ? dans la requête SQL (syntaxe standard). Si la requête comporte plusieurs caractères ?, plusieurs appels à SQL FIXER PARAMETRE seront nécessaires. Les valeurs des objets 4D seront affectées séquentiellement dans la requête, dans l'ordre d'exécution des commandes.

Exemples

(1) Cet exemple permet d'exécuter une requête SQL faisant directement appel à des variables 4D associées :

```
C_TEXTE(MonTexte)
C_ENTIER LONG(MonEntierLong)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MonTexte>>,
                                                                    <<MonEntierLong>>)"
```

```
Boucle (vCounter;1;10)
    MonTexte:="Texte"+Chaine(vCounter)
    MonEntierLong:=vCounter
    SQL EXECUTER(SQLStmt)
    SQL ANNULER CHARGEMENT
Fin de boucle
SQL LOGOUT
```

(2) Même exemple que le précédent, mais en utilisant la commande SQL FIXER PARAMETRE :

```
C_TEXTE(MonTexte)
C_ENTIER LONG(MonEntierLong)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (?,?)"
Boucle (vCounter;1;10)
    MonTexte:="Texte"+Chaine(vCounter)
    MonEntierLong:=vCounter
    SQL FIXER PARAMETRE(MonTexte;SQL Paramètre entrée )
    SQL FIXER PARAMETRE(MonEntierLong;SQL Paramètre entrée)
    SQL EXECUTER(SQLStmt)
    SQL ANNULER CHARGEMENT
Fin de boucle
SQL LOGOUT
```

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL IMPORTER

SQL IMPORTER (tableSource{; projet{; *}})

Paramètre	Type		Description
tableSource	Chaîne	→	Nom de la table dans la source de données ODBC
projet	BLOB	→	Contenu du projet d'importation
		←	Nouveau contenu du projet (si * est passé)
*	*	→	Affichage de la boîte de dialogue d'importation et mise à jour du projet

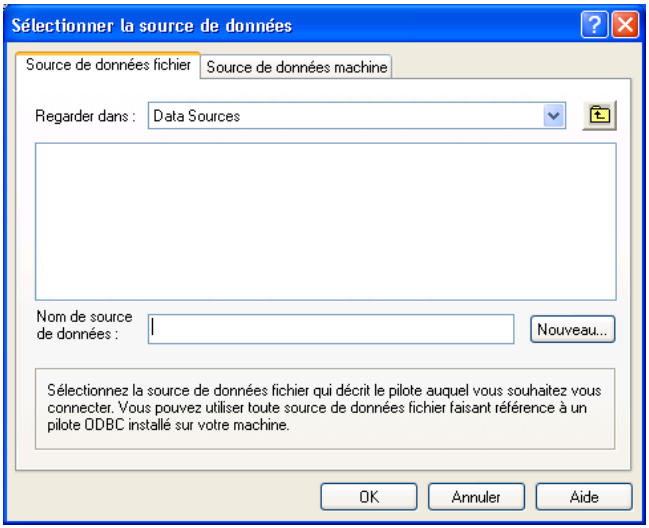
Description

La commande SQL IMPORTER permet d'importer des données depuis la table tableSource d'une source ODBC externe. Les paramètres de connexion (nom de la source, utilisateur et mot de passe) sont inclus dans le BLOB projet.

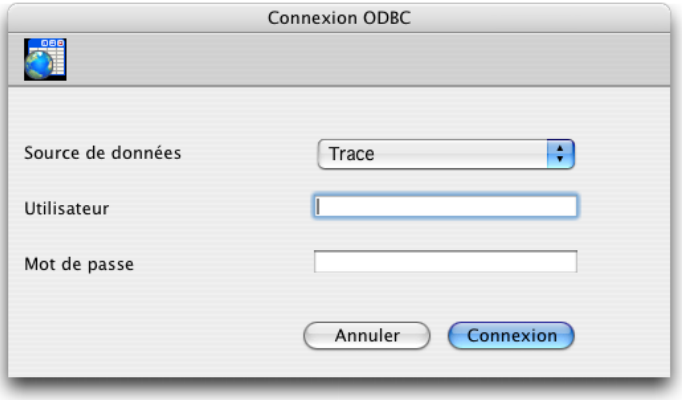
Notes :
• Le *projet* contient tous les paramètres de l'importation, notamment la source de données et les tables et champs d'arrivée. Vous définissez ces paramètres dans la boîte de dialogue d'importation ODBC, puis vous pouvez éventuellement les sauvegarder dans un fichier sur disque. Pour plus d'informations, reportez-vous au manuel *Mode Développement*.
• Les projets générés dans la boîte de dialogue d'importation ODBC ne sont pas compatibles avec les commandes ou la boîte de dialogue d'importation standard de 4D.
• Cette commande ne peut pas être utilisée dans le cadre d'une connexion avec le moteur SQL interne de 4D.

Si vous ne passez pas le paramètre facultatif projet, SQL IMPORTER provoque l'affichage de la boîte de dialogue de sélection de la source de données ODBC :

Windows



Mac OS



Une fois la source sélectionnée, la boîte de dialogue d'importation ODBC de 4D apparaît, permettant à l'utilisateur de configurer l'opération. Si l'utilisateur clique sur le bouton **Annuler** dans l'une des boîtes de dialogue, l'exécution est stoppée et la variable système OK prend la valeur 0.

Si vous passez dans le paramètre projet un BLOB contenant un projet d'import ODBC valide, l'importation s'effectue directement, sans intervention de l'utilisateur. Pour cela, il vous suffit de charger un projet préalablement sauvegardé sur disque dans le champ ou la variable BLOB que vous passez dans le paramètre projet, à l'aide de la commande DOCUMENT VERS BLOB.

Vous pouvez également utiliser la commande SQL IMPORTER avec un paramètre projet vide et le paramètre optionnel *, puis stocker le paramètre projet dans un champ BLOB (cf. ci-dessous). Cette solution permet, d'une part, de le conserver avec le fichier de données et d'autre part d'éviter la phase de chargement dans un BLOB depuis le disque.

Note : Reportez-vous à la commande EXPORTER DONNEES pour un exemple de définition de projet vide.

Le paramètre optionnel *, s'il est spécifié, provoque l'affichage de la boîte de dialogue d'importation de données de 4D avec les paramétrages éventuellement définis dans le projet. Ce fonctionnement permet d'utiliser un projet prédéfini tout en ayant la possibilité de modifier un ou plusieurs paramètres. En outre, dans ce cas le paramètre projet contient, après la fermeture de la boîte de dialogue, les paramètres du "nouveau" projet. Vous pouvez alors le stocker dans un champ BLOB, dans un fichier disque, etc.

Référence

SQL EXPORTER.

Variables et ensembles système

Si l'utilisateur clique sur le bouton **Annuler** dans une des boîtes de dialogue (de sélection de source de données ou de paramétrage d'import), la variable système OK prend la valeur 0. Si l'importation se déroule correctement, la variable système OK prend la valeur 1.

SQL LIRE DERNIERE ERREUR

version 2004

SQL LIRE DERNIERE ERREUR (errCode; errTexte; errODBC; errSQLServer)

Paramètre	Type		Description
errCode	Entier long	←	Code de l'erreur
errTexte	Texte	←	Texte de l'erreur
errODBC	Texte	←	Code d'erreur ODBC
errSQLServer	Entier long	←	Code d'erreur native serveur SQL

Description

La commande SQL LIRE DERNIERE ERREUR retourne des informations relatives à la dernière erreur rencontrée lors de l'exécution d'une commande ODBC. L'erreur peut provenir de l'application 4D, du réseau, de la source ODBC, etc.

Cette commande doit généralement être appelée dans le contexte d'une méthode de gestion des erreurs installée à l'aide de la commande APPELER SUR ERREUR.

- Le paramètre errCode retourne le code de l'erreur.
- Le paramètre errTexte retourne le libellé de l'erreur.

Les deux derniers paramètres ne sont remplis que si l'erreur provient de la source ODBC. Dans le cas contraire, ils sont retournés vides.
- Le paramètre errODBC retourne le code d'erreur ODBC (*SQL state*).
- Le paramètre errSQLServer retourne le code de l'erreur native du serveur SQL.

Référence

APPELER SUR ERREUR, LIRE PILE DERNIERE ERREUR.

SQL LIRE OPTION

SQL LIRE OPTION (option; valeur)

Paramètre	Type		Description
option	Entier long	→	Numéro d'option
valeur	Entier long	←	Valeur de l'option

Description

La commande SQL LIRE OPTION retourne la valeur courante de l'option passée dans le paramètre option.

Pour plus d'informations sur les différentes options et leurs valeurs associées, reportez-vous à la description de la commande SQL FIXER OPTION.

Référence

SQL FIXER OPTION.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK retourne 1, sinon elle retourne 0.

SQL LOGIN{(source; utilisateur; motDePasse; *)}

Paramètre	Type		Description
source	Chaîne	→	• Nom de la base externe ou • Adresse IP de la base externe ou • Nom de la source de données dans le gestionnaire ODBC ou • "" pour afficher le dialogue de sélection
utilisateur	Chaîne	→	Nom d'utilisateur enregistré dans la source de données
motDePasse	Chaîne	→	Mot de passe de l'utilisateur
*	*	→	Appliquer à Debut SQL/Fin SQL Si omis : ne pas appliquer (base locale), si passé : appliquer

Description

La commande SQL LOGIN vous permet d'ouvrir une connexion avec une source de données SQL, définie dans le paramètre source. Elle désigne la cible des requêtes SQL exécutées ultérieurement dans l'application :
• via la commande SQL EXECUTER,
• via le code placé à l'intérieur des balises Debut SQL / Fin SQL (si le paramètre * est passé).

La source de données SQL peut être soit :
• une base 4D Server externe à laquelle vous accédez directement,
• une source ODBC externe,
• le moteur SQL interne.

Vous pouvez passer dans source l'une des valeurs suivantes : une *adresse IP*, un *nom de publication de base 4D*, un *nom de source de données ODBC*, une *chaîne vide* ou la constante *SQL_INTERNAL*.

• *adresse IP*
Syntaxe : IP:<Adresse IP>{:<PortTCP>}
Dans ce cas, la commande ouvre une connexion directe avec la base 4D Server exécutée sur l'ordinateur ayant l'adresse IP définie. Sur l'ordinateur "cible", le serveur SQL doit être lancé. Si vous passez un numéro de port TCP, il doit avoir été spécifié comme port de publication du serveur SQL dans la base "cible". Si vous ne passez pas de numéro de port TCP, le port par défaut sera utilisé (19812). Le numéro de port TCP du serveur SQL peut être modifié dans la page "SQL/Configuration" des Préférences de l'application.
Reportez-vous aux exemples 1 et 2.

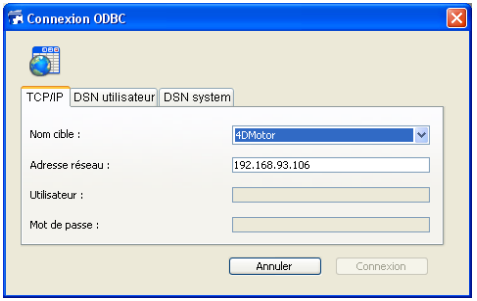
• *nom de publication de base 4D*
Syntaxe : 4D:<Nom_de_Publication>
Dans ce cas, la commande ouvre une connexion directe avec la base 4D Server dont le nom de publication sur le réseau correspond au nom spécifié. Le nom de publication réseau d'une base est défini dans la page "Client-Serveur/Configuration" des Préférences de l'application.
Reportez-vous à l'exemple 4.

Note : Le numéro de port TCP du serveur SQL 4D cible (qui publie la base 4D) et le numéro de port TCP du serveur SQL de l'application 4D ouvrant la connexion doivent être identiques.

• *nom de source de données ODBC valide*
Syntaxe : ODBC:<Ma_DSN> ou <Ma_DSN>
Dans ce cas, le paramètre source contient le nom de la source de données telle qu'elle a été définie dans le gestionnaire du pilote ODBC.

Note : Par compatibilité avec les versions précédentes de 4D, il est possible d'omettre le préfixe "ODBC:". Toutefois pour des raisons de lisibilité du code il est conseillé d'utiliser ce préfixe.
Reportez-vous à l'exemple 4.

• *chaîne vide*
Syntaxe : ""
Dans ce cas la commande provoque l'affichage de la boîte de dialogue de connexion, permettant de désigner manuellement la source de données à laquelle se connecter :



Cette boîte de dialogue comporte plusieurs pages. La page TCP/IP se compose des éléments suivants :
*** Nom cible** : ce menu est construit à l'aide de deux listes :
- la liste des bases ouvertes récemment en connexion directe. Le mécanisme de mise à jour de cette liste est identique à celui de l'application 4D, à la différence près que le dossier contenant les fichiers .4DLink est nommé "Favorites SQL v11" au lieu de "Favorites v11".
- la liste des applications 4D Server dont le serveur SQL est lancé et dont le port TCP pour les connexions SQL est égal à celui de l'application source. Cette liste est mise à jour dynamiquement à chaque nouvel appel de la commande SQL LOGIN sans le paramètre source. Le caractère "^" placé devant un nom de base indique que la connexion est effectuée en mode sécurisé via SSL.

*** Adresse réseau** : cette zone affiche l'adresse IP et éventuellement le port TCP de la base sélectionnée dans le menu Nom cible. Vous pouvez également saisir dans cette zone une adresse IP puis cliquer sur le bouton Connexion afin de vous connecter à la base 4D Server correspondante. Vous pouvez également spécifier le port TCP, en saisissant deux points (:) puis le numéro du port à la suite de l'adresse. Par exemple : 192.168.93.105:19855
*** Utilisateur** et **Mot de passe** : ces zones permettent de saisir les identifiants de la connexion.
***** Les pages **DSN utilisateur** et **DSN système** affichent respectivement la liste des sources de données ODBC utilisateur et système définies dans le gestionnaire ODBC de la machine. Ces pages permettent de sélectionner une source de données et de saisir des identifiants afin d'ouvrir une connexion avec une source ODBC externe.
Si la connexion est établie, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est générée. Cette erreur peut être interceptée via une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

• constante *SQL_INTERNAL*
Syntaxe : SQL_INTERNAL
Dans ce cas, la commande redirige les requêtes SQL suivantes vers le moteur SQL interne de la base.

Le paramètre utilisateur contient le nom de l'utilisateur autorisé à se connecter à la source de données externe. Par exemple, avec Oracle®, ce nom d'utilisateur peut être "Scott".

Le paramètre motDePasse contient le mot de passe de l'utilisateur autorisé à se connecter. Par exemple, avec Oracle®, ce mot de passe peut être "tiger".

Note : Dans le cas d'une connexion directe, si vous passez des chaînes vides dans les paramètres utilisateur et motDePasse, la connexion ne sera acceptée que si les mots de passe 4D ne sont pas activés dans la base cible. Sinon, la connexion est refusée.

Le paramètre facultatif * permet de changer la cible du code SQL exécuté au sein des balises Debut SQL/Fin SQL. Si vous ne passez pas ce paramètre, le code placé dans les balises Debut SQL/Fin SQL sera toujours adressé au moteur SQL interne de 4D, sans tenir compte du paramétrage défini par la commande SQL LOGIN. Si vous passez ce paramètre, le code SQL exécuté au sein des balises Debut SQL/Fin SQL sera adressé à la source définie par la commande.

Pour refermer la connexion courante et libérer la mémoire, il suffit d'exécuter la commande SQL LOGOUT. Toutes les requêtes SQL sont alors dirigées vers le moteur SQL interne de la base.
Si vous appelez une nouvelle fois SQL LOGIN sans avoir refermé explicitement la connexion courante, elle est automatiquement refermée.

Tous les paramètres sont facultatifs. Si aucun paramètre n'est passé, la commande provoquera l'affichage de la boîte de dialogue de connexion ODBC, permettant de désigner manuellement la source de données à laquelle se connecter.

La portée de cette commande est le process. Autrement dit, si vous souhaitez ouvrir deux connexions distinctes, vous devez créer deux process et ouvrir chaque connexion dans chaque process.

Exemples

(1) Cette instruction provoque l'affichage de la boîte de dialogue du gestionnaire ODBC :

```
SQL LOGIN
```

(2) Ouverture d'une connexion via le protocole ODBC avec la source de données externe "MonOracle". Les requêtes SQL exécutées via la commande SQL EXECUTER et les requêtes incluses dans les balises Debut SQL/Fin SQL seront redirigées vers cette connexion :

```
SQL LOGIN("ODBC:MonOracle";"Scott";"tiger";*)
```

(3) Ouverture d'une connexion avec le moteur SQL interne de 4D :

```
SQL LOGIN(SQL_INTERNAL;$utilisateur;$motdepasse)
```

(4) Ouverture d'une connexion directe avec l'application 4D Server v11 SQL exécutée sur le poste ayant l'adresse IP 192.168.45.34 et répondant sur le port TCP par défaut. Les requêtes SQL exécutées via la commande SQL EXECUTER seront redirigées vers cette connexion, les requêtes incluses dans les balises Debut SQL/Fin SQL ne seront pas redirigées.

```
SQL LOGIN("IP:192.168.45.34";"John";"azerty")
```

(5) Ouverture d'une connexion directe avec l'application 4D Server v11 SQL exécutée sur le poste ayant l'adresse IP 192.168.45.34 et répondant sur le port TCP 20150. Les requêtes SQL exécutées via la commande SQL EXECUTER et les requêtes incluses dans les balises Debut SQL/Fin SQL seront redirigées vers cette connexion.

```
SQL LOGIN("IP:192.168.45.34:20150";"John";"azerty";*)
```

(6) Ouverture d'une connexion directe avec l'application 4D Server v11 SQL qui publie sur le réseau local une base dont le nom de publication est "DB_Compta". Le port TCP utilisé pour le serveur SQL des deux bases (défini dans la page "SQL/Configuratio"n des Préférences) doit être identique (19812 par défaut). Les requêtes SQL exécutées via la commande SQL EXECUTER seront redirigées vers cette connexion, les requêtes incluses dans les balises Debut SQL/Fin SQL ne seront pas redirigées.

```
SQL LOGIN ("4D:DB_Compta";"John";"azerty")
```

(7) Cet exemple illustre les possibilités de connexion offertes par la commande SQL LOGIN :

```
TABLEAU TEXTE (30;aNames)
TABLEAU ENTIER LONG(aAges;0)
SQL LOGIN("ODBC:MonORACLE";"Marc";"azerty")
Si (OK=1)
      `La requête suivante sera redirigée vers la base ORACLE externe
   SQL EXECUTER("SELECT Name, Age FROM PERSONS";aNames; aAges)
      `La requête suivante sera dirigée vers la base 4D locale
   Debut SQL
      SELECT Name, Age
      FROM PERSONS
      INTO :aNames, :aAges;
   Fin SQL
      `La commande SQL LOGIN suivante referme la connexion courante
      `avec la base externe ORACLE et ouvre une nouvelle connexion avec
      `une base externe MySQL
   SQL LOGIN ("ODBC:MySQL";"Jean";"qwerty";*)
   Si (OK=1)
         `La requête suivante sera redirigée vers la base MySQL externe
      SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames; aAges)
         `La requête suivante sera aussi redirigée vers la base MySQL externe
      Debut SQL
         SELECT Name, Age
         FROM PERSONS
         INTO :aNames, :aAges;
      Fin SQL
      SQL LOGOUT
         `La requête suivante sera dirigée vers la base 4D locale
      Debut SQL
         SELECT Name, Age
         FROM PERSONS
         INTO :aNames, :aAges;
      Fin SQL
   Fin de si
Fin de si
```

Référence

Debut SQL, Fin SQL, SQL LOGOUT.

Variables et ensembles système

Si la connexion est correctement établie, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

SQL LOGOUT

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande SQL LOGOUT referme la connexion avec une source ODBC ouverte dans le process courant (le cas échéant). S'il n'y a pas de connexion ODBC ouverte, la commande ne fait rien.

Référence

SQL LOGIN.

Variables et ensembles système

Si la connexion a été correctement refermée, la variable système OK prend la valeur 1, sinon elle prend la valeur 0. Vous pouvez intercepter les éventuelles erreurs à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

UTILISER BASE INTERNE

version 11.3 (Modifiée)

Note de compatibilité : Cette commande est remplacée par la commande SQL LOGOUT à compter de la version 11.3 de 4D. Elle est conservée pour des raisons de compatibilité uniquement et ne sera pas maintenue dans les prochaines versions du programme.

UTILISER BASE INTERNE

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Variables et ensembles système

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est générée.

UTILISER BASE EXTERNE

version 11.3 (Modifiée)

Note de compatibilité : Cette commande est remplacée par la commande SQL LOGIN à compter de la version 11.3 de 4D. Elle est conservée pour des raisons de compatibilité uniquement et ne sera pas maintenue dans les prochaines versions du programme.

UTILISER BASE EXTERNE (nomSource{; utilisateur; motDePasse})

Paramètre	Type		Description
nomSource	Chaîne	→	Nom de la source de données ODBC à laquelle se connecter
utilisateur	Chaîne	→	Nom d'utilisateur
motDePasse	Chaîne	→	Mot de passe de l'utilisateur

Variables et ensembles système

Si la commande est correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est générée.

Valeur champ Null

Valeur champ Null (unChamp) → Booléen

Paramètre	Type		Description
unChamp	Champ	→	Champ à évaluer
Résultat	Booléen	←	Vrai = le champ est NULL, Faux = le champ n'est pas NULL

Description

La commande Valeur champ Null retourne Vrai si le champ désigné par le paramètre unChamp contient la valeur NULL, et Faux sinon.

La valeur NULL est exploitée par le moteur SQL de 4D. Pour plus d'informations, reportez-vous au *Manuel de référence SQL 4D*.

Référence

FIXER VALEUR CHAMP NULL.

57

Table

Table du formulaire courant

Table

version 11 (Modifiée)

Table du formulaire courant → Pointeur

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Pointeur	←	Pointeur vers la table à laquelle appartient le formulaire actuellement affiché

Description

La fonction Table du formulaire courant retourne un pointeur vers la table à laquelle appartient le formulaire affiché à l'écran ou imprimé dans le process courant.

La fonction retourne Nil dans les cas suivants :
• il n'y a pas de formulaire affiché ou en cours d'impression dans le process courant,
• le formulaire courant est un formulaire projet.

Si plusieurs fenêtres sont ouvertes simultanément dans le process courant (ce qui signifie que la dernière fenêtre ouverte est la fenêtre courante active), la fonction retourne un pointeur vers la table du formulaire affiché dans la fenêtre active.

Si le formulaire affiché est le formulaire détaillé d'une zone de sous-formulaire (c'est-à-dire que pendant la saisie de données, l'utilisateur a double-cliqué sur un enregistrement ou un sous-enregistrement dans une zone de sous-formulaire "double-cliquable"), la fonction retourne :
• un pointeur vers la table de la zone de sous-formulaire, si cette dernière affiche une table.
• un pointeur non significatif si la zone de sous-formulaire affiche une sous-table.

Exemple

Dans votre application, vous utilisez la convention suivante : au moment de l'affichage d'un enregistrement, la variable vsEnrCourant, si elle est présente dans un formulaire, affiche "Nouvel enregistrement" si vous créez un nouvel enregistrement, ou par exemple "56 parmi 5200" si vous ouvrez le 56e enregistrement d'une sélection en comportant 5200. Pour cela, vous pouvez créer une fois la variable vsEnrCourant et lui associer la méthode objet décrite ci-dessous, puis la copier et la coller dans tous les formulaires que vous voulez :

```
      ` Méthode objet de la variable non saisissable vsEnrCourant
Au cas ou
   : (Evenement formulaire =Sur chargement)
      C_ALPHA (31;vsEnrCourant)
      C_POINTEUR ($vpTableParente)
      C_ENTIER LONG ($vlNumEnr)
      $vpTableParente:=Table du formulaire courant
      $vlNumEnr:=Numero enregistrement ($vpTableParente->)
      Au cas ou
         : ($vlNumEnr=-3)
            vsEnrCourant:="Nouvel enregistrement"
         : ($vlNumEnr=-1)
            vsEnrCourant:="Pas d'enregistrement"
         : ($vlNumEnr>=0)
            vsEnrCourant:=Chaine (Numero dans selection ($vpTableParente->))
                  +" parmi "+Chaine (Enregistrements trouves($vpTableParente->))
      Fin de cas
Fin de cas
```

Référence

DIALOGUE, FORMULAIRE ENTREE, FORMULAIRE SORTIE, IMPRIMER SELECTION.

TABLE PAR DEFAUT

version 5

TABLE PAR DEFAUT (laTable)

Paramètre	Type		Description
laTable	Table	→	Table à définir comme table par défaut

Description

TABLE PAR DEFAUT définit laTable comme la table par défaut pour le process courant.

Un process n'a pas de table par défaut tant que la commande TABLE PAR DEFAUT n'a pas été exécutée. Après qu'une table par défaut ait été désignée, toute commande pour laquelle le paramètre laTable n'a pas été défini s'appliquera à la table par défaut. Considérez par exemple l'instruction suivante :

```
FORMULAIRE ENTREE ([maTable];"Formulaire")
```

Si [maTable] a préalablement été définie comme table par défaut, la même instruction pourrait s'écrire :

```
FORMULAIRE ENTREE ("Formulaire")
```

Une des raisons pour lesquelles vous pouvez définir une table par défaut est l'écriture de code qui ne soit pas lié à une table. Cela permet au même code d'être appliqué à différentes tables. Vous pouvez aussi utiliser des pointeurs vers des tables pour écrire du code non lié aux tables. Pour plus d'informations sur cette technique, reportez-vous à la description de la commande Nom de la table.

TABLE PAR DEFAUT ne permet pas d'omettre les noms de tables lorsque vous vous référez à des champs. Par exemple :

```
[MaTable]MonChamp := "Une chaîne"   ` Correct
```

ne peut pas s'écrire :

```
TABLE PAR DEFAUT ([MaTable])
MonChamp := "Une chaîne"    ` Incorrect
```

... simplement parce qu'une table par défaut a été définie.

Toutefois, vous pouvez omettre le nom de la table lorsque vous vous référez à des champs dans des triggers, des formulaires et des objets appartenant à la table.

Dans 4D, toutes les tables sont “ouvertes” et prêtes à être utilisées. TABLE PAR DEFAUT n'ouvre pas de table, ne définit pas de table courante et ne prépare pas de table pour la saisie ou l'affichage. TABLE PAR DEFAUT est simplement une facilité de programmation proposée pour accélérer la saisie du code et le rendre plus facile à lire.

Conseil : Bien que l'appel de TABLE PAR DEFAUT et l'omission du nom de la table rendent le code plus lisible, la plupart des programmeurs estiment que l'utilisation de cette commande apporte plus d'inconvénients que d'avantages.

Exemple

L'exemple suivant présente la même méthode avec et sans la commande TABLE PAR DEFAUT. Le code est une boucle souvent utilisée pour créer de nouveaux enregistrements dans une base. Les commandes FORMULAIRE ENTREE et AJOUTER ENREGISTREMENT nécessitent le nom d'une table comme premier paramètre :

```
FORMULAIRE ENTREE ([Clients]; "Ajout Enrg")
Repeter
    AJOUTER ENREGISTREMENT ([Clients])
Jusque (OK = 0)
```

Voici le résultat lorsqu'une table par défaut est définie :

```
TABLE PAR DEFAUT ([Clients])
FORMULAIRE ENTREE ("Ajout Enrg")
Repeter
    AJOUTER ENREGISTREMENT
Jusque (OK = 0)
```

Référence

PAS DE TABLE PAR DEFAUT, Table par defaut courante.

Table par defaut courante

Table par defaut courante → Pointeur

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Pointeur	←	Pointeur vers la table par défaut

Description

Table par defaut courante retourne un pointeur vers la table qui a été passée au dernier appel de la commande TABLE PAR DEFAUT pour le process courant.

Exemple

La ligne de code suivante inscrit le nom de la table courante par défaut dans le titre de la fenêtre :

```
CHANGER TITRE FENETRE(Nom de la table(Table par defaut courante))
```

Référence

Nom de la table, Table, TABLE PAR DEFAUT.

PAS DE TABLE PAR DEFAUT

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande PAS DE TABLE PAR DEFAUT permet d'annuler l'effet de la commande TABLE PAR DEFAUT. Après l'exécution de cette commande, il n'y a plus de table par défaut définie pour le process. Si la commande TABLE PAR DEFAUT n'avait pas été appelée au préalable, cette commande ne fait rien.

Cette commande est liée à l'utilisation de formulaires projets (formulaires non liés à des tables) : la plupart des commandes relatives aux formulaires (hors formulaires utilisateurs) acceptent un paramètre facultatif de type table comme premier paramètre. C'est par exemple le cas des commandes LIRE PARAMETRE FORMULAIRE, Creer fenetre formulaire ou DIALOGUE. Comme un formulaire projet et un formulaire table peuvent avoir le même nom, ce paramètre permet de déterminer le formulaire à utiliser : passez le paramètre lorsque vous souhaitez adresser un formulaire table et ne le passez pas dans le cas d'un formulaire projet.
Dans une base contenant un formulaire projet nommé "LeForm" et un formulaire table du même nom pour la table [Table1] :

```
DIALOGUE([Table1];"LeForm") `4D utilise le formulaire table
DIALOGUE("LeForm") `4D utilise le formulaire projet
```

Ce principe est toutefois caduc lorsque la commande TABLE PAR DEFAUT a été exécutée et que la base contient un formulaire projet et un formulaire table du même nom. En effet, dans ce cas 4D utilisera le formulaire table de la table par défaut, même si le paramètre laTable n'est pas passé. Dans ce cas, pour permettre l'utilisation de formulaires projet, il suffit d'exécuter la commande PAS DE TABLE PAR DEFAUT.

Exemple

Dans une base contenant un formulaire projet nommé "LeForm"et un formulaire table du même nom pour la table [Table1] :

```
TABLE PAR DEFAUT([Table1])
DIALOGUE("LeForm") `4D utilise le formulaire table
PAS DE TABLE PAR DEFAUT
DIALOGUE("LeForm") `4D utilise le formulaire projet
```

Référence

TABLE PAR DEFAUT.

58

Tableaux

Un **tableau** est une série ordonnée de variables de même type. Chaque variable est appelée un **élément** du tableau. La **taille** d'un tableau est le nombre d'éléments qu'il contient. La taille du tableau doit être définie au moment de sa création ; vous pouvez ensuite la modifier aussi souvent que nécessaire en ajoutant, insérant, ou supprimant des éléments, ou en appelant de nouveau la commande que vous avez utilisée pour créer le tableau.

Vous créez un tableau au moyen de l'une des commandes de déclaration de tableau. Pour plus d'informations, reportez-vous à la section Créer des tableaux.

Les éléments sont numérotés de **1 à N**, où N est la taille du tableau. Un tableau a toujours un **élément zéro**, auquel vous pouvez accéder tout comme vous accédez à n'importe quel autre élément du tableau, mais cet élément n'est pas affiché lorsqu'un tableau est utilisé dans un formulaire. Rien ne vous empêche cependant de l'utiliser avec le langage. Pour plus d'informations sur l'élément zéro, reportez-vous à la section Utiliser l'élément zéro d'un tableau.

Les tableaux sont des variables 4D. Comme toute variable, un tableau a une portée et suit les règles du langage 4D, bien qu'il existe quelques différences spécifiques. Pour plus d'informations, reportez-vous aux sections Les tableaux et le langage 4D et Tableaux et Pointeurs.

Les tableaux sont des objets du langage : vous pouvez créer et utiliser des tableaux qui n'apparaîtront jamais à l'écran. Mais les tableaux sont également des objets d'interface utilisateur. Pour plus d'informations sur l'interaction entre les tableaux et les objets de formulaire, reportez-vous aux sections Tableaux et objets de formulaire et Zones de défilement groupées.

Les tableaux doivent être utilisés pour manipuler une certaine quantité de données pendant une période brève. En contrepartie, comme les tableaux résident en mémoire, ils sont d'une utilisation rapide et facile. Pour plus d'informations, reportez-vous à la section Tableaux et mémoire.

Vous créez un tableau au moyen de l'une des commandes de déclaration de tableau décrites dans ce chapitre. Voici la liste des commandes de déclaration de tableau :

Commande	**Crée ou redimensionne un tableau de** :
TABLEAU ENTIER	Entiers (sur 2 octets)
TABLEAU ENTIER LONG	Entiers (sur 4 octets)*
TABLEAU REEL	Réels
TABLEAU TEXTE	Textes (jusqu'à 2 Go de texte par élément)**
TABLEAU ALPHA	(*obsolète en mode Unicode*)**
TABLEAU DATE	Dates
TABLEAU BOOLEEN	Booléens
TABLEAU IMAGE	Images
TABLEAU POINTEUR	Pointeurs

Chaque commande de déclaration de tableau peut créer ou redimensionner des tableaux à une ou à deux dimensions. Pour plus d'informations sur les tableaux à deux dimensions, reportez-vous à la section Tableaux à deux dimensions.

(*) Les tableaux de type Entier long vous permettent de manipuler les données de type Heure. Pour afficher dans un formulaire un tableau sous forme d'heures, appliquez à l'objet affichant le tableau le format d'affichage &/x, où x représente le numéro d'ordre du format souhaité dans la liste de formatage des heures. Par exemple, &/4 correspond au format Heure:Minute.

(**) Les tableaux Texte et les tableaux Alpha manipulent le même type d'éléments : des caractères.
• **En mode Unicode** (mode par défaut pour les bases de données créées à compter de la version 11 de 4D), il n'y a aucune différence entre les tableaux Texte et les tableaux Alpha. Le paramètre longueurChaîne de la commande TABLEAU ALPHA est ignoré. Dans ce contexte, il est conseillé d'utiliser des tableaux Texte. La commande TABLEAU ALPHA est conservée pour des raisons de compatibilité uniquement.
• **En mode compatibilité** (non Unicode), les commandes TABLEAU TEXTE et TABLEAU ALPHA diffèrent :
- dans un tableau Texte, chaque élément est de longueur variable et stocke ses caractères dans des parties différentes de la mémoire. Chaque élément d'un tableau Texte peut contenir jusqu'à 32 000 caractères.

- dans un tableau Alpha, tous les éléments ont la même longueur fixe (la longueur définie par le paramètre longueurChaîne lors de la création du tableau). Tous les éléments sont stockés les uns à la suite des autres dans la même partie de la mémoire, ce qui rend les tableaux Alpha plus rapides que les tableaux Texte. Cependant, un élément de tableau Alpha ne peut dépasser 255 caractères.
Pour plus d'informations, reportez-vous à la section Codes ASCII.

Cette ligne de code crée (déclare) un tableau d'entiers de 10 éléments :

```
TABLEAU ENTIER(aiUnTableau;10)
```

Ensuite, cette ligne de code redimensionne le même tableau à 20 éléments :

```
TABLEAU ENTIER(aiUnTableau;20)
```

Enfin, cette ligne de code redimensionne le même tableau à 0 élément :

```
TABLEAU ENTIER(aiUnTableau;0)
```

Vous référencez les éléments d'un tableau en utilisant des accolades ({…}). Un nombre entre accolades donne accès à l'adresse d'un élément particulier. Ce nombre est appelé **numéro de l'élément**. L'exemple ci-dessous place cinq noms dans le tableau nommé atNoms et les affiche ensuite dans une fenêtre d'alerte :

```
TABLEAU TEXTE (atNoms;5)
atNoms{1}  := "Richard"
atNoms{2}  := "Sarah"
atNoms{3}  := "Pierre"
atNoms{4}  := "Martine"
atNoms{5}  := "Jean"
Boucle ($vlElem;1;5)
    ALERTE ("L'élément #"+Chaine($vlElem)+" est égal à: "+atNoms{$vlElem} )
Fin de boucle
```

Notez la syntaxe atNoms{$vlElem}. Au lieu de spécifier un nombre littéral comme atNoms{3}, vous pouvez utiliser une variable numérique indiquant à quel élément d'un tableau vous accédez.

Si vous utilisez les itérations permises par les structures répétitives (Boucle...Fin de boucle, Repeter...Jusque ou Tant que...Fin tant que), vous pouvez accéder à tout ou partie des éléments d'un tableau avec très peu de code.

Les tableaux et les autres commandes du langage 4D

Il existe d'autres commandes 4D qui permettent de créer ou de manipuler des tableaux. En particulier :

• Pour travailler avec des tableaux et des sélections d'enregistrements, utilisez les commandes SELECTION LIMITEE VERS TABLEAU, SELECTION VERS TABLEAU, TABLEAU VERS SELECTION et VALEURS DISTINCTES.

• Les objets de type List box sont basés sur les tableaux ; plusieurs des commandes du thème “List box” manipulent des tableaux, par exemple INSERER LIGNE LISTBOX.

• Vous pouvez créer des graphes à partir de valeurs stockées dans des tables et des tableaux. Pour plus d'informations, reportez-vous à la commande GRAPHE.

• Bien que la version 6 apporte un jeu complet de commandes fonctionnant avec les listes hiérarchiques, les commandes ENUMERATION VERS TABLEAU et TABLEAU VERS ENUMERATION (de la version précédente de 4D) ont été conservées pour des raisons de compatibilité.

• De nombreuses commandes peuvent construire des tableaux en un appel, par exemple LISTE DES POLICES, LISTE FENETRES, LISTE DES VOLUMES, LISTE DES DOSSIERS, LISTE DES DOCUMENTS, LIRE CORRESPONDANCE PORT SERIE, SAX LIRE ELEMENT XML, etc.

Référence

Présentation des tableaux, TABLEAU ALPHA, TABLEAU BOOLEEN, TABLEAU DATE, TABLEAU ENTIER, TABLEAU ENTIER LONG, TABLEAU IMAGE, TABLEAU POINTEUR, TABLEAU REEL, TABLEAU TEXTE, Tableaux à deux dimensions.

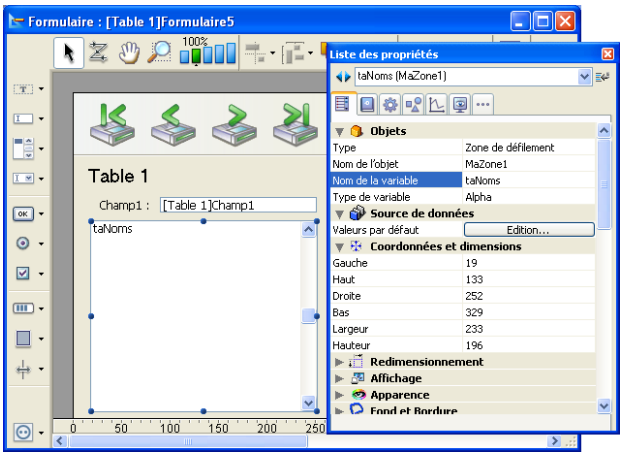
Les tableaux sont des objets de langage — vous pouvez créer et utiliser des tableaux qui n'apparaîtront jamais à l'écran. Cependant, les tableaux sont aussi des objets d'interface utilisateur. Voici les types d'**objets de formulaire** gérés par des tableaux :

- Pop-up/Liste déroulante
- Combo Box
- Zone de défilement
- Onglet
- List box

Si vous pouvez prédéfinir ces objets dans l'éditeur de formulaires en utilisant le bouton des valeurs par défaut de la Liste des propriétés (hormis les List box), vous pouvez également les définir par programmation, en utilisant les commandes de tableaux. Dans les deux cas, l'objet formulaire est géré par un tableau, créé par vous ou par 4D.

En utilisant ces objets, vous pouvez détecter quel élément de l'objet a été sélectionné (ou a reçu un clic souris) en testant l'**élément sélectionné** du tableau. Inversement, vous pouvez sélectionner un élément de l'objet en désignant l'élément de tableau correspondant.

Quand un tableau est utilisé pour gérer un objet de formulaire, il a une nature double ; il est à la fois un objet de langage et un objet d'interface utilisateur. Par exemple, créez un formulaire, dans lequel vous placez une zone de défilement :



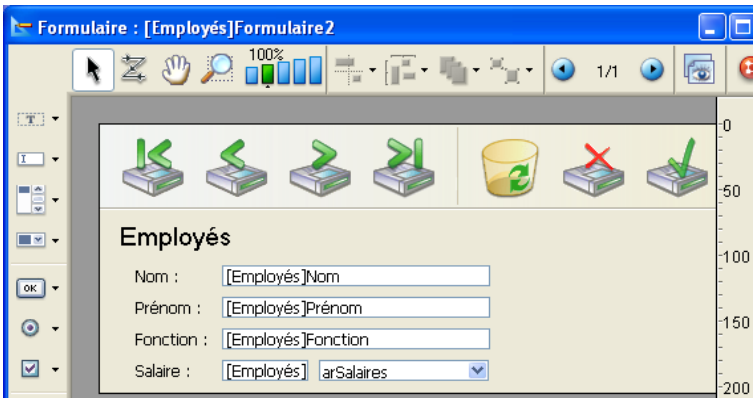
Le nom de la variable associée, ici taNoms, est le nom du tableau que vous utilisez pour créer et gérer la zone de défilement.

Notes :
• Vous ne pouvez pas afficher des tableaux à deux dimensions ni des tableaux de pointeurs.
• La gestion des objets de type **List box** (pouvant contenir plusieurs tableaux) revêt de nombreux aspects particuliers. Ces spécificités sont traitées dans la section Gestion programmée des objets de type List box.

Exemple : création d'une liste déroulante

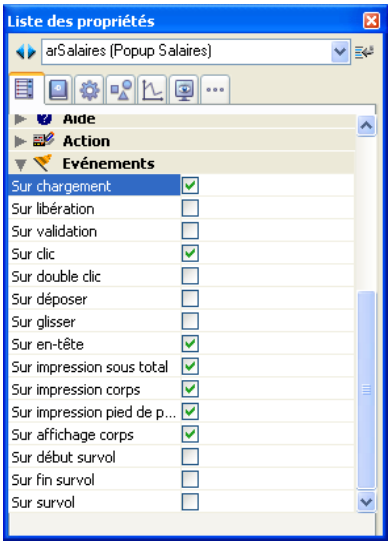
L'exemple suivant montre comment remplir un tableau et l'afficher dans une liste déroulante. Un tableau arSalaires est créé au moyen de la commande TABLEAU REEL. Il contient tous les salaires des personnes dans une entreprise américaine. Lorsque l'utilisateur sélectionne un élément dans la liste déroulante, le champ [Personnel]Salaire reçoit la valeur choisie.

Création de la liste déroulante arSalaires dans un formulaire
Créez une liste déroulante et nommez-la arSalaires. Le nom de la liste déroulante doit être le même que celui du tableau.

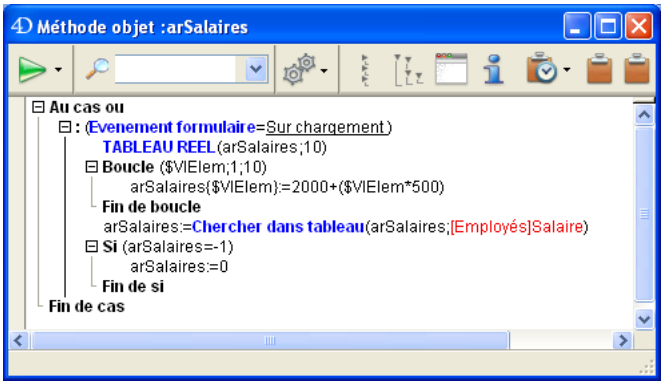


Initialisation du tableau

Initialisez le tableau arSalaires en spécifiant l'événement Sur chargement pour l'objet. Pour cela, n'oubliez pas d'activer cet événement dans la Liste des propriétés, comme illustré ci-dessous :



Cliquez sur le bouton **Méthode objet...** et écrivez la méthode suivante :



Les lignes :

```
TABLEAU REEL(arSalaires;10)
Boucle($vlElem;1;10)
    arSalaires{$vlElem}:=2000+($vlElem*500)
Fin de boucle
```

... créent le tableau numérique 2500, 3000... 7000, correspondant aux salaires annuels allant de $30 000 à $84 000, avant impôts.

Les lignes :

```
arSalaires:=Chercher dans tableau(arSalaires;[Employés]Salaire)
Si (arSalaires=-1)
    arSalaires:=0
Fin de si
```

... gèrent à la fois la création d'un nouvel enregistrement et la modification d'un enregistrement existant.

• Si vous créez un nouvel enregistrement, le champ [Employés]Salaire est initialement égal à zéro. Dans ce cas, Chercher dans tableau ne trouve pas la valeur dans le tableau et retourne -1. Le test Si (arSalaires=-1) remet arSalaires à zéro, indiquant qu'aucun élément n'est sélectionné dans la liste déroulante.
• Si vous modifiez un enregistrement existant, Chercher dans tableau récupère la valeur dans le tableau et affecte à l'élément sélectionné de la liste déroulante la valeur courante du champ. Si la valeur pour un employé n'est pas dans la liste, le test Si (arSalaires=-1) désélectionne tous les éléments de la liste.

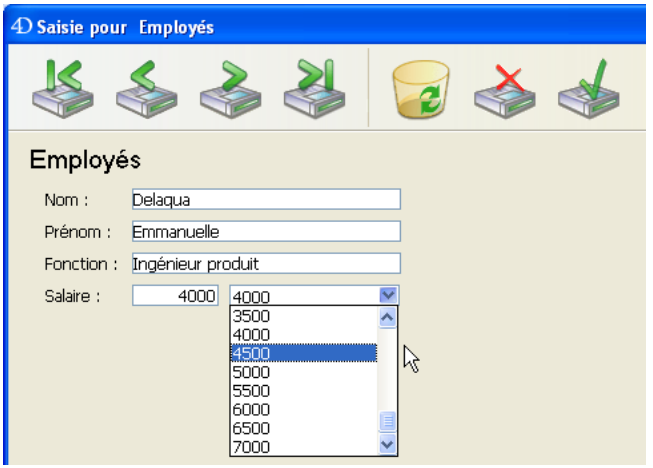
Note : Pour plus d'informations concernant l'**élément de tableau sélectionné**, lisez les paragraphes suivants.

Assignation de la valeur sélectionnée au champ [Employés]Salaire
Pour reporter la valeur sélectionnée dans la liste déroulante arSalaires, il vous suffit de gérer l'événement Sur clic de l'objet. Le numéro de l'élément sélectionné est la valeur du tableau arSalaires. En conséquence, l'expression arSalaires{arSalaires} retourne la valeur choisie dans la liste déroulante.

Complétez ainsi la méthode de l'objet arSalaires :

```
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU REEL(arSalaires;10)
        Boucle($vlElem;1;10)
            arSalaires{$vlElem} :=2000+($vlElem*500)
        Fin de boucle
        arSalaires:=Chercher dans tableau(arSalaires;[Employés]Salaire)
        Si (arSalaires=-1)
            arSalaires:=0
        Fin de si
    : (Evenement formulaire=Sur clic)
        [Employés]Salaire:=arSalaires{arSalaires}
Fin de cas
```

En mode Utilisation ou Menus créés, la liste déroulante se présente comme suit :



Les paragraphes suivants décrivent les opérations élémentaires que vous pouvez effectuer sur les tableaux lorsque vous les utilisez comme objets de formulaire.

Obtenir la taille d'un tableau

Vous pouvez obtenir la taille courante d'un tableau au moyen de la commande Taille tableau. Si on reprend l'exemple précédent, la ligne de code qui suit affichera 5:

```
ALERTE ("La taille du tableau taNoms est: "+Chaine(Taille tableau(taNoms)))
```

Trier (réordonner) les éléments du tableau

Vous pouvez réordonner les éléments d'un tableau au moyen de la commande TRIER TABLEAU ou de plusieurs tableaux à l'aide de la commande TABLEAU MULTI TRI. Si on reprend l'exemple précédent, et étant entendu que le tableau est affiché comme une liste déroulante, vous pourrez voir ceci :

(a) Sur la gauche, la liste telle qu'elle se présente initialement.

(b) Au centre, la liste telle qu'elle se présente après l'exécution de la ligne de code suivante :

```
TRIER TABLEAU(taNoms;>)
```

(c) Sur la droite, la liste telle qu'elle se présente après l'exécution de la ligne de code suivante :

```
TRIER TABLEAU(taNoms;<)
```

Ajouter et supprimer des éléments

Vous pouvez ajouter, insérer, ou supprimer des éléments de tableau au moyen des commandes AJOUTER A TABLEAU, INSERER DANS TABLEAU et SUPPRIMER DANS TABLEAU.

Gestion des clics dans un tableau : test de l'élément sélectionné

Si on reprend l'exemple précédent, étant entendu que le tableau est affiché en tant que liste déroulante, vous pouvez gérer les clics souris de la manière suivante :

```
    ` Méthode objet liste déroulante taNoms
Au cas ou
    : (Evenement formulaire=Sur chargement)
            ` Initialiser le tableau (comme vu précédemment)
        TABLEAU TEXTE (taNoms;5)
            ` ...
    : (Evenement formulaire=Sur libération)
            ` Nous n'avons plus besoin du tableau
        EFFACER VARIABLE(taNoms)

    : (Evenement formulaire=Sur clic)
        Si (taNoms#0)
            vtInfo:="Vous avez cliqué sur : "+taNoms{taNoms}
        Fin de si
    : (Evenement formulaire=Sur double clic)
        Si (taNoms#0)
            ALERTE ("Vous avez double-cliqué sur : "+taNoms{taNoms})
        Fin de si
Fin de cas
```

Note : Les événements doivent avoir été activés dans les propriétés de l'objet.

Alors que la syntaxe taNoms{$vlElem} vous permet de travailler sur un élément particulier du tableau, la syntaxe taNoms retourne le numéro de l'élément sélectionné dans le tableau. Ainsi, la syntaxe taNoms{taNoms} signifie “la valeur de l'élément sélectionné dans le tableau taNoms.” Si aucun élément n'est sélectionné, taNoms est égal à 0 (zéro). Le test Si (taNoms#0) détecte si un élément est effectivement sélectionné ou non.

Désigner l'élément sélectionné

Vous pouvez changer par programmation l'élément sélectionné en assignant une valeur au tableau. Exemples:

```
    ` Sélectionner le premier élément (si le tableau n'est pas vide)
taNoms:=1
```

```
    ` Sélectionner le dernier élément (si le tableau n'est pas vide)
taNoms:=Taille tableau(taNoms)

    ` Déselectionner l'élément sélectionné (s'il y en a un), aucun élément n'est alors
                                                                                sélectionné
taNoms:=0

Si ((0<taNoms)&(taNoms<Taille tableau(taNoms))
        ` Si possible, sélectionner l'élément suivant l'élément sélectionné
    taNoms:=taNoms+1
Fin de si

Si (1<taNoms)
        ` Si possible, sélectionner l'élément précédent l'élément sélectionné
    taNoms:=taNoms-1
Fin de si
```

Recherche d'une valeur dans le tableau

La commande Chercher dans tableau recherche une valeur particulière dans un tableau. Si nous reprenons l'exemple précédent, voici le code qui sélectionnera l'élément dont la valeur est “Richard”, si c'est ce que vous saisissez dans la boîte de dialogue de demande :

```
$vsNom:=Demander("Saisissez un prénom :")
Si (OK=1)
    $vlElem:=Chercher dans tableau (taNoms;$vsNom)
    Si ($vlElem>0)
        taNoms:=$vlElem
    Sinon
        ALERTE ("Il n'y a pas de "+$vsName+" dans cette liste de prénoms.")
    Fin de si
Fin de si
```

Les pop-up menus, listes déroulantes, zones de défilement et les onglets peuvent généralement être gérés de la même manière. Bien entendu, aucun code supplémentaire n'est nécessaire pour le redessinement des objets à l'écran à chaque fois que vous modifiez la valeur d'un élément, en ajoutez ou en supprimez.

Note : Pour créer et utiliser des onglets avec des icônes ainsi que des onglets activés ou désactivés, vous devez utiliser une liste hiérarchique comme objet de gestion associé à l'onglet. Pour plus d'informations, reportez-vous à l'exemple de la commande Nouvelle liste.

Gestion des combo boxes

Alors que vous pouvez gérer au moyen des algorithmes décrits dans la section précédente les pop-up menus, les listes déroulantes, les zones de défilement et les onglets, vous devez gérer les combo boxes différemment.

Une combo box est en réalité une zone de texte saisissable à laquelle est rattachée une liste de valeurs prédéfinies (les éléments d'un tableau). L'utilisateur peut choisir une valeur dans cette liste, et ensuite éditer le texte. En conséquence, pour une combo box, la notion d'élément sélectionné ne s'applique pas.

Avec les combo boxes, il n'y a jamais d'élément sélectionné. A chaque fois que l'utilisateur sélectionne une des valeurs attachées à la zone, cette valeur est placée dans l'élément zéro du tableau. Ensuite, si l'utilisateur modifie le texte, la valeur modifiée est aussi placée dans cet élément zéro. Exemple :

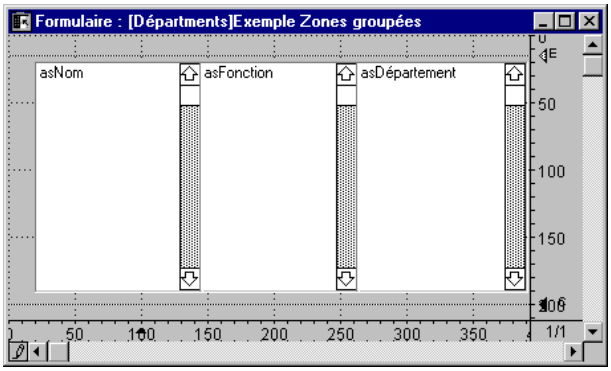
```
    ` Méthode objet Combo Box asCouleurs
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TABLEAU ALPHA(31;asCouleurs;3)
        asCouleurs{1} :="Bleu"
        asCouleurs{2} :="Blanc"
        asCouleurs{3} :="Rouge"
    : (Evenement formulaire=Sur clic)
        Si (asCouleurs{0} #"")
                ` L'objet change automatiquement de valeur
                ` L'utilisation de l'événement Sur clic avec une Combo Box
                ` n'est requise que pour prendre en compte des actions supplémentaires
        Fin de si
    : (Evenement formulaire=Sur données modifiées)
            ` Chercher dans tableau ignore l'élément 0, et donc retourne -1 ou >0
        Si (Chercher dans tableau(asCouleurs;asCouleurs{0} )<0)
                ` La valeur saisie n'est pas une des valeurs attachées à l'objet
                ` Ajouter la valeur à la liste pour la prochaine fois
            AJOUTER A TABLEAU(asCouleurs;asCouleurs{0})
        Sinon
                ` La valeur entrée est une des valeurs attachées à l'objet
        Fin de si
Fin de cas
```

Référence

Présentation des tableaux, Zones de défilement groupées.

Note de compatibilité : Les zones de défilement groupées sont toujours utilisables dans 4D, toutefois à compter de la version 2004 du programme elles peuvent être avantageusement remplacées par des objets de type List box. Pour plus d'informations sur ce point, reportez-vous à la section Présentation des List box.

Vous pouvez grouper des zones de défilement pour l'affichage dans un formulaire. Lorsque plusieurs zones de défilement sont groupées, elles se comportent comme une seule zone de défilement. Chaque zone peut disposer de ses propres police et taille de caractères ; cependant, nous vous recommandons d'utiliser la même hauteur de police dans chaque colonne. Lors de la saisie de données, seule la zone de défilement située au premier plan affiche une barre de défilement. Voici trois zones de défilement groupées en mode Développement, dans l'éditeur de formulaires :

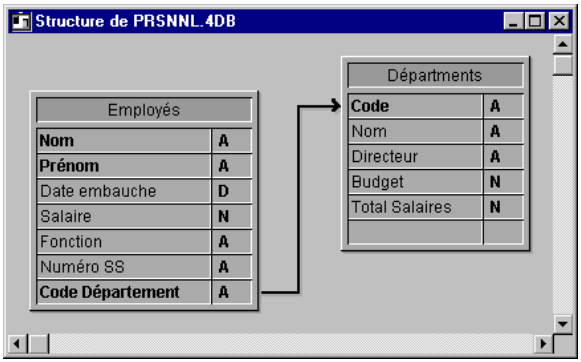


Voici quelques conseils pour la création de zones de défilement groupées :
• Assurez-vous que tous les tableaux ont la même taille (nombre d'éléments).
• Utilisez la même taille de caractères dans chaque zone.
• Veillez à ce toutes les zones aient la même hauteur.
• Alignez toutes les zones sur le haut.
• Assurez-vous que les zones ne se chevauchent pas.
• Assurez-vous que la zone de droite est au premier plan, car la barre de défilement apparaît sur la zone de premier plan.
• Groupez les zones (au moyen de la commande de menu **Grouper**) pour qu'elles fonctionnent comme s'il s'agissait d'une seule zone de défilement.

Cette méthode projet remplit les trois tableaux et les affiche à l'écran :

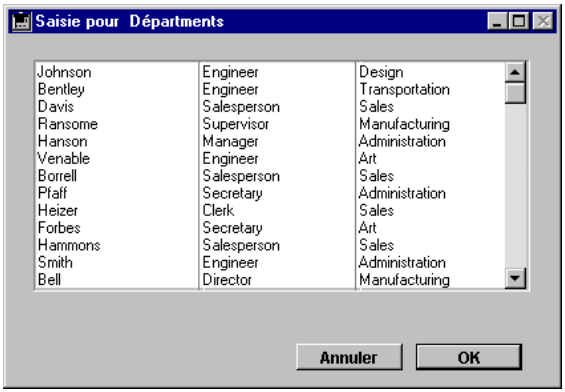
```
TOUT SELECTIONNER(Employés)
SELECTION VERS TABLEAU([Employés]Nom de Famille;asNom;[Employés]Fonction;
                                                    asFonction;[Départements]Nom;asDépartement)
DIALOGUE([Départements];"Exemple Zones groupées")
```

Cette méthode utilise les données des champs de la table [Employés] et de la table [Départements]. Voici ces tables :



Note : La table [Départements] peut être utilisée à condition qu'il y ait un lien automatique allant de [Employés] à [Départements].

Voici le résultat :

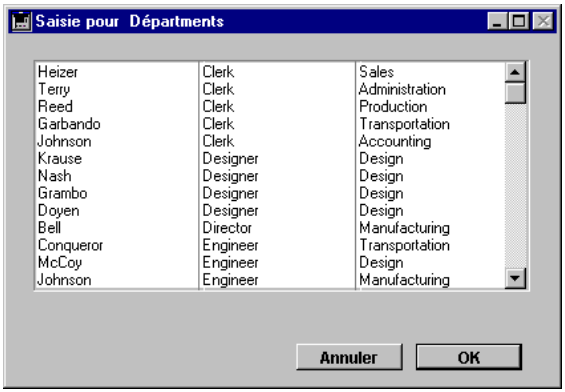


Notez qu'une seule barre de défilement est affichée. Quand l'utilisateur clique sur une ligne, les trois zones sont sélectionnées simultanément. La variable associée à chaque zone de défilement prend le numéro de la ligne ayant reçu le clic souris ; seule la méthode objet de la zone cliquée est exécutée. Par exemple, si l'utilisateur clique sur le nom “Bentley”, asNom, asFonction, et asDépartement prennent la valeur 2, mais seule la méthode objet de asNom est exécutée.

Les tableaux peuvent être triés au moyen de la commande TRIER TABLEAU. Par exemple :

```
TRIER TABLEAU(asFonction;asNom;asDépartement;>)
```

Voici le résultat du tri :



Notez que les tableaux ont été triés sur la base du premier argument de la commande TRIER TABLEAU ; les deux autres tableaux ont été désignés pour assurer la synchronisation entre les lignes. La commande TRIER TABLEAU trie toujours les tableaux (s'il y en a plusieurs) sur les valeurs du premier argument et assure la synchronisation des autres tableaux.

Note : La commande TRIER TABLEAU ne permet pas de trier les tableaux sur plusieurs niveaux. Pour afficher un tableau similaire au tableau ci-dessus tout en ayant un tri sur plusieurs niveaux (par exemple par département, puis par fonction, puis par nom), utilisez un sous-formulaire dans lequel vous affichez la table et utilisez la commande TRIER.

Référence

Présentation des tableaux, Tableaux et objets de formulaire.

Les tableaux sont des variables 4D. Comme toute variable, un tableau a une portée (une aire d'action) et suit les règles du langage 4D, à certaines différences près.

Tableaux locaux, process et interprocess

Vous pouvez créer des tableaux locaux, process ou interprocess, par exemple :

```
    ` Ceci crée un tableau local de 100 valeurs entières sur 2 octets
TABLEAU ENTIER ($aiCodes;100)
    ` Ceci crée un tableau process de 100 valeurs entières sur 2 octets
TABLEAU ENTIER (aiCodes;100)
    ` Ceci crée un tableau interprocess de 100 valeurs entières sur 2 octets
TABLEAU ENTIER (<>aiCodes;100)
```

La portée de ces tableaux est identique à celle des autres variables locales, process et interprocess.

Tableaux locaux
Vous déclarez un tableau local lorsque son nom commence par le signe dollar ($).

La portée d'un tableau local est la méthode dans laquelle il est créé. Le tableau est effacé lorsque la méthode est terminée. Des tableaux locaux de même nom peuvent avoir des types différents dans deux méthodes différentes, car ce sont en réalité des variables différentes n'ayant pas la même portée.

Lorsque vous créez un tableau local dans une méthode formulaire ou objet, ou dans une méthode projet appelée comme sous-routine par l'un des deux types de méthodes précédents, le tableau est créé puis effacé à chaque fois que la méthode formulaire ou la méthode objet est utilisée. En d'autres termes, le tableau est créé puis effacé pour chaque événement formulaire. Par conséquent, vous ne pouvez pas utiliser de tableaux locaux dans des formulaires, ni pour l'affichage ni pour l'impression.

Comme dans le cas des variables locales, il est préférable d'utiliser les tableaux locaux à chaque fois que c'est possible. Vous réduisez ainsi la quantité de mémoire nécessaire pour votre application.

Tableaux process
Vous déclarez un tableau process lorsque le nom du tableau débute par une simple lettre.

La portée d'un tableau process est le process dans lequel il a été créé. Le tableau est effacé lorsque le process se termine ou est tué. Un tableau process dispose d'une instance créée automatiquement pour chaque process. Par conséquent, le tableau est du même type pour tous les process. En revanche, son contenu est particulier à chaque process.

Tableaux interprocess
Vous déclarez un tableau interprocess lorsque son nom commence par <> (sous Windows et Mac OS) ou par le signe "diamant" (Mac OS uniquement — Option+v sur un clavier français).

La portée d'un tableau interprocess est la totalité des process pendant la session de travail. Il convient de ne les utiliser que pour partager des données ou transférer des informations entre les process.

Astuce : Quand vous savez d'avance qu'un tableau interprocess sera utilisé par plusieurs process, ce qui peut provoquer des conflits, protégez l'accès à ce tableau par un sémaphore. Pour plus d'informations, reportez-vous à l'exemple de la commande Semaphore .

Note : Vous pouvez utiliser des tableaux process et interprocess dans des formulaires pour créer des objets de formulaire tels que des zones de défilement, des listes déroulantes, etc.

Passer un tableau comme paramètre

Vous pouvez passer un tableau en tant que paramètre à une commande 4D ou à une routine d'un plug-in 4D. Cependant, vous ne pouvez pas passer un tableau comme paramètre à une méthode utilisateur. La solution dans ce cas est de passer un pointeur vers le tableau comme paramètre à cette méthode. Pour plus de détails, reportez-vous à la section Tableaux et pointeurs.

Affecter un tableau à un autre tableau

Contrairement à ce que vous pouvez faire avec des variables de type Texte ou Chaîne, vous ne pouvez pas affecter un tableau à un autre tableau. Pour copier (affecter) un tableau à un autre, utilisez la fonction COPIER TABLEAU.

Référence

Présentation des tableaux, Tableaux et pointeurs.

Vous pouvez passer un tableau comme paramètre à une commande 4D ou à une routine d'un Plug-in 4D. Cependant, vous ne pouvez pas passer un tableau comme paramètre dans une méthode utilisateur. La solution consiste à passer un pointeur vers le tableau comme paramètre de la méthode.

Vous pouvez passer des tableaux interprocess, process ou locaux comme paramètres.

Voici quelques exemples.

• Prenons le cas suivant :

```
Si ((0<atNoms)&(atNoms<Taille tableau(atNoms))
        ` Si possible, sélectionner l'élément suivant l'élément sélectionné
    atNoms:=atNoms+1
Fin de si
```

Si vous avez besoin de faire la même chose pour 50 tableaux différents, vous pouvez vous éviter d'écrire 50 fois la même chose, en utilisant la méthode projet suivante:

```
    ` Méthode projet SELECTIONNER ELEMENT SUIVANT
    ` SELECTIONNER ELEMENT SUIVANT ( Pointeur )
    ` SELECTIONNER ELEMENT SUIVANT ( -> Tableau )

    C_POINTEUR ($1)

    Si ((0<$1->)&($1-><Taille tableau($1->))
        $1->:=$1->+1 ` Si possible, sélectionner l'élément suivant l'élément sélectionné
    Fin de si
```

Ensuite, vous pouvez écrire :

```
SELECTIONNER ELEMENT SUIVANT (->atNoms)
    ` ...
SELECTIONNER ELEMENT SUIVANT (->asCodesPostaux)
    ` ...
SELECTIONNER ELEMENT SUIVANT (->alEnrgsIDs)
        ` ... et ainsi de suite.
```

• La méthode projet suivante retourne la somme de tous les éléments d'un tableau numérique (Entier, Entier long, ou Réel) :

```
    ` Somme Tableau
    ` Somme Tableau ( Pointeur )
    ` Somme Tableau ( -> Tableau )

C_REEL ($0)

$0:=0
Boucle ($vlElem;1;Taille tableau($1->))
    $0:=$0+$1->{$vlElem}
Fin de boucle
```

Ensuite, vous pouvez écrire :

```
$vlSomme:=Somme Tableau (->arSalaires)
    ` ...
$vlSomme:=Somme Tableau (->aiDefectCounts)
    ` ..
$vlSomme:=Somme Tableau (->alPopulations)
```

• La méthode projet qui suit met une majuscule à tous les éléments d'un tableau Alpha ou Texte :

```
    ` MAJUSCULE TABLEAU
    ` MAJUSCULE TABLEAU ( Pointeur )
    ` MAJUSCULE TABLEAU ( -> Tableau )

Boucle ($vlElem;1;Taille tableau($1->))
    Si ($1->{$vlElem} #"")
        $1->{$vlElem}:=Majusc($1->{$vlElem} [[1]])+Minusc(Sous chaine($1->{$vlElem};2))
    Fin de si
Fin de boucle
```

Ensuite, vous pouvez écrire :

```
MAJUSCULE TABLEAU (->atSujets )
    ` ...
MAJUSCULE TABLEAU (->asNomsFamille )
```

La combinaison de tableaux, pointeurs et de boucles telles que Boucle...Fin de boucle vous permet d'écrire un grand nombre de petites méthodes projet très utiles pour gérer les tableaux.

Référence

Les tableaux et le langage 4D, Présentation des tableaux.

Utiliser l'élément zéro d'un tableau

Un tableau a toujours un élément zéro. Même si l'élément zéro n'est pas affiché lorsqu'un tableau est utilisé pour remplir un objet de formulaire, vous pouvez l'utiliser sans réserve dans le langage.

Un exemple possible d'utilisation de l'élément zéro est le cas de la combo box examiné dans la section Tableaux et objets de formulaire.

Voici deux autres exemples :

(1) Si vous voulez exécuter une action seulement lorsque vous cliquez sur un élément autre que l'élément préalablement sélectionné, vous pouvez garder la trace de chaque élément sélectionné. Une façon de le faire est d'utiliser une variable process dans laquelle vous conservez le numéro de l'élément sélectionné. Une autre manière consiste à utiliser l'élément zéro du tableau :

```
      ` Méthode objet zone de défilement atNoms
Au cas ou
   : (Evenement formulaire=Sur chargement)
         ` Initialisont le tableau
      TABLEAU TEXTE (atNoms;5)
         ` ...
         ` Initialiser l'élément zéro avec le numéro
         ` de l'élément courant sélectionné sous sa forme alphanumérique
         ` Ici vous commencez sans élément sélectionné
      atNoms{0} :="0"

   : (Evenement formulaire=Sur libération)
         ` Nous n'avons plus besoin du tableau
      EFFACER VARIABLE(atNoms)
```

```
    : (Evenement formulaire=Sur clic)
        Si (atNoms#0)
            Si (atNoms#Num(atNoms{0} ))
                vtInfo:="Vous avez cliqué sur : "+atNoms{atNoms} +
                                                    " qui n'était pas précédemment sélectionné."
                atNoms{0} :=Chaine(atNoms)
            Fin de si
        Fin de si
    : (Evenement formulaire=Sur double clic)
        Si (atNoms#0)
            ALERTE ("Vous avez double-cliqué sur : "+atNoms{atNoms})
        Fin de si
Fin de cas
```

(2) En mode compatibilité ASCII, lorsque vous envoyez des caractères vers un document ou le port série, ou bien en recevez, vous avez la possibilité de filtrer les codes ASCII entre les plates-formes et systèmes dont les tables ASCII diffèrent — via les commandes UTILISER FILTRE, Mac vers ISO, ISO vers Mac, Mac vers Windows et Windows vers Mac.

Dans certains cas, vous pouvez souhaiter contrôler intégralement la traduction des codes ASCII. Pour cela, vous pouvez utiliser un tableau d'entiers de 255 éléments, dans lequel le Nième élément est la traduction ASCII du caractère dont le code ASCII d'origine est N. Par exemple, si le code ASCII 187 doit devenir 156, vous écrivez <>tiFiltreAsciiExport{187}:=156 et <>tiFiltreAsciiImport{156}:=187 dans la méthode qui initialise les tableaux interprocess utilisés dans la base. Vous pouvez alors envoyer une suite de caractères en utilisant une méthode projet telle que celle-ci :

```
    ` X ENVOYER PAQUET ( Texte { ; Heure }  )
Boucle ($vlChar;1;Longueur($1))
    $1[[vlChar]]:=Caractere(<>tiFiltreAsciiExport{Code de caractere($1[[vlChar]])} )
Fin de boucle
Si (Nombre de parametres>=2)
    ENVOYER PAQUET ($2;$1)
Sinon
    ENVOYER PAQUET ($1)
Fin de si

    ` X Recevoir paquet ( Texte { ; Heure }  ) -> Texte
Si (Nombre de parametres>=2)
    RECEVOIR PAQUET ($2;$1)
```

```
Sinon
    RECEVOIR PAQUET ($1)
Fin de si
$0:=$1
Boucle ($vlChar;1;Longueur($1))
    $0[[vlChar]]:=Caractere(<>tiFiltreAsciiImport{Code de caractere($0[[vlChar]])} )
Fin de boucle
```

Dans cet exemple, si une suite de caractères contenant des caractères NULL (code ASCII zéro) est envoyée ou reçue, l'élément zéro des tableaux <>tiFiltreAsciiExport et <>tiFiltreAsciiImport jouera son rôle comme n'importe lequel des 255 éléments du tableau.

Référence

Présentation des tableaux.

Tableaux à deux dimensions

Chaque commande de déclaration de tableau permet de créer ou de redimensionner des tableaux à une ou à deux dimensions. Exemple :

```
TABLEAU TEXTE (atTopics;100;50) ` Créer un tableau texte composé de 100 lignes de 50 colonnes
```

Les tableaux à deux dimensions sont essentiellement des objets de langage ; vous ne pouvez ni les afficher ni les imprimer.

Dans l'exemple prédédent :
- atTopics est un tableau à deux dimensions.
- atTopics{8} {5} est le 5e élément (5e colonne...) de la 8e ligne.
- atTopics{20} est la 20e ligne et est elle-même un tableau à une dimension.
- Taille tableau(atTopics) retourne 100, qui est le nombre de lignes
- Taille tableau(atTopics{17}) retourne 50, qui est le nombre de colonnes de la 17e ligne

Dans l'exemple suivant, un pointeur vers chaque champ de chaque table de la base est stocké dans un tableau à deux dimensions :

```
C_ENTIER LONG($vlDerniereTable;$vlDernierChamp)
C_ENTIER LONG($vlNumeroChamp)
    ` Créer autant de lignes (vides et sans colonnes) qu'il y a de tables
$vlDerniereTable:=Lire numero derniere table
TABLEAU POINTEUR(<>apChamps;$vlDerniereTable;0)
    `Tableau 2D avec N lignes et zéro colonnes
    ` Pour chaque table
Boucle ($vlTable;1;$vlDerniereTable)
    Si (Est un numero de table valide($vlTable))
        $vlDernierChamp:=Lire numero dernier champ($vlTable)
            ` Donner la valeur des éléments
        $vlNumeroColonne:=0
        Boucle ($vlChamp;1;$vlDernierChamp)
            Si (Est un numero de champ valide($vlTable;$vlChamp))
                $vlNumeroColonne:=$vlNumeroColonne+1
                    `Insère une colonne dans la ligne de la table en cours
```

```
                    INSERER DANS TABLEAU(<>apChamps{$vlTable};$vlNumeroColonne;1)
                        `Affecte la "celulle" avec le pointeur
                    <>apChamps{$vlTable}{$vlNumeroColonne}:=Champ($vlTable;$vlChamp)
                Fin de si
            Fin de boucle
        Fin de si
    Fin de boucle
```

Dans la mesure où le tableau à deux dimensions a été initialisé, vous pouvez obtenir ainsi les pointeurs vers les champs d'une table de votre choix :

```
    ` Obtenir les pointeurs vers les champs pour la table affichée à l'écran:
COPIER TABLEAU (<>apChamps{Table(Table du formulaire courant)};
                                                    $apMesChampsdeTravail)
    ` Initialiser les champs booléens et date
Boucle ($vlElem;1;Taille tableau($apMesChampsdeTravail))
    Au cas ou
        : (Type($apMesChampsdeTravail{$vlElem} ->)=Is Date)
            $apMesChampsdeTravail{$vlElem} ->:=Date du jour
        : (Type($apMesChampsdeTravail{$vlElem} ->)=Is Boolean)
            $apMesChampsdeTravail{$vlElem} ->:=Vrai
    Fin de cas
Fin de boucle
```

Note : Comme le montre cet exemple, les lignes des tableaux à deux dimensions peuvent être ou non de la même taille, indifféremment.

Référence

Présentation des tableaux.

A la différence des données que vous stockez sur disque lorsque vous utilisez des tables ou des enregistrements, un tableau **réside toujours en mémoire dans son intégralité**.

Par exemple, si tous les codes postaux américains étaient saisis dans une table [Codes Postaux], celle-ci contiendrait environ 100 000 enregistrements. De plus, cette table comporterait plusieurs champs : le code postal lui-même ainsi que la ville, le comté et l'état correspondants. Si vous ne sélectionnez que les codes postaux de Californie, 4D crée la sélection d'enregistrements correspondante à l'intérieur de la table [Codes Postaux], et ensuite ne charge les enregistrements que lorsque vous en avez besoin (par exemple, pour les afficher ou les imprimer). En d'autres termes, vous travaillez avec une série ordonnée de valeurs (du même type pour chaque champ) partiellement chargée du disque en mémoire.

Procéder de la même manière avec les tableaux serait laborieux, pour les raisons suivantes :

• Pour maintenir les quatre types d'information (code postal, ville, comté, état), vous auriez besoin de quatre grands tableaux en mémoire.
• Comme un tableau réside en mémoire dans son intégralité, vous seriez obligé de garder tous les codes postaux en mémoire pendant toute la session de travail, même si les données n'étaient pas utilisées en permanence.
• Toujours parce qu'un tableau réside en mémoire dans son intégralité, les quatre tableaux devraient être chargés ou sauvegardés sur le disque à chaque fois que vous démarreriez ou quitteriez l'application, quand bien même les données ne seraient d'aucune utilité pour la session de travail.

Conclusion : Les tableaux ont pour rôle de manipuler une certaine quantité de données pendant une période brève. En contrepartie, comme ils résident en mémoire, ils sont d'une utilisation rapide et facile.

Cependant, dans certaines circonstances, vous pouvez avoir besoin de tableaux contenant des centaines ou des milliers d'éléments. Voici les formules à appliquer pour calculer la quantité de mémoire utilisée pour chaque type de tableau:

Type de Tableau	Calcul de la quantité de mémoire en octets
Booléen	(31+nombre d'éléments)/8
Date	(1+nombre d'éléments) * 6
Alpha	(1+nombre d'éléments) * longueur déclarée (+1 si impair, +2 si pair)
Entier	(1+nombre d'éléments) * 2

Entier long	(1+nombre d'éléments) * 4
Image	(1+nombre d'éléments) * 4 + somme de la taille de chaque image
Pointeur	(1+nombre d'éléments) * 16
Réel	(1+nombre d'éléments) * 8
Texte	(1+nombre d'éléments) * 6 + somme de la taille de chaque texte
Deux dimensions	(1+nombre d'éléments) * 12 + somme de la taille de chaque tableau

Note : Quelques octets supplémentaires sont requis pour le repérage de l'élément, le nombre d'éléments et le tableau lui-même.

Lorsque vous travaillez avec de très grands tableaux, la meilleure façon de gérer d'éventuels problèmes de saturation de la mémoire est d'accompagner la création de tableau d'une méthode projet APPELER SUR ERREUR. Exemple :

```
    ` Vous allez lancer une opération batch fonctionnant toute la nuit
    ` qui requiert la création de grands tableaux. Pour éviter
    ` des erreurs en pleine nuit, créez les tableaux au début de
    ` l'opération et testez les erreurs au même moment :
gError:=0 ` Initialisation
    ` Installation de la méthode de gestion d'erreurs
APPELER SUR ERREUR ("GESTION ERREUR")
TABLEAU ALPHA (63;asCeTableau;50000) ` Environ 3125 Ko
TABLEAU REEL (arCetAutreTableau;50000) ` 488 Ko
APPELER SUR ERREUR ("") ` Il n'est plus nécessaire d'intercepter les erreurs
Si (gError=0)
    ` Les tableaux ont pu être créés
    ` poursuivons l'opération
Sinon
    ALERTE ("Cette opération requiert davantage de mémoire !")
Fin de si
    ` Dans tous les cas, nous n'avons plus besoin des tableaux
EFFACER VARIABLE (asCeTableau)
EFFACER VARIABLE (arCetAutreTableau)
```

La méthode projet GESTION ERREUR est la suivante :

```
    ` Méthode projet GESTION ERREUR
gError:=Error ` Retourner le code d'erreur
```

Référence

APPELER SUR ERREUR, Présentation des tableaux.

AJOUTER A TABLEAU

AJOUTER A TABLEAU (tableau; valeur)

Paramètre	Type		Description
tableau	Tableau	→	Tableau auquel ajouter une valeur
valeur	Expression	→	Valeur à ajouter au tableau

Description

La commande AJOUTER A TABLEAU ajoute une nouvelle ligne à la fin du tableau et lui affecte la valeur passée dans le paramètre valeur. En mode interprété, si le tableau n'a pas été défini au préalable, la commande le crée et lui attribue un type en fonction de celui de valeur.

Cette commande fonctionne avec tous les types de tableaux : chaîne, numérique, booléen, date, pointeur et image.

Le type de valeur doit correspondre au type du tableau, sinon l'erreur de syntaxe 54 "Les arguments sont incompatibles" est générée. Les combinaisons suivantes sont toutefois possibles :
• un tableau de type chaîne (Texte ou Alpha) accepte toute valeur de type Texte ou Alpha.
• un tableau de type numérique (Entier, Entier long ou Réel) accepte toute valeur de type Entier, Entier long, Numérique ou Heure.

Exemple

Le code suivant :

```
INSERER DANS TABLEAU($montableau;Taille tableau($montableau)+1)
$montableau{Taille tableau($montableau)}:=$mavaleur
```

... peut être remplacé par :

```
AJOUTER A TABLEAU($montableau;$mavaleur)
```

Référence

INSERER DANS TABLEAU, SUPPRIMER DANS TABLEAU.

Chercher dans tableau

Chercher dans tableau (tableau; valeur{; début}) → Numérique

Paramètre	Type		Description
tableau	Tableau	→	Tableau dans lequel effectuer la recherche
valeur	Expression	→	Valeur de même type à rechercher dans le tableau
début	Numérique	→	Elément à partir duquel commencer la recherche
Résultat	Numérique	←	Numéro du premier élément trouvé correspondant à valeur

Description

Chercher dans tableau retourne le numéro du premier élément de tableau qui correspond à valeur.

Chercher dans tableau peut être utilisé avec des tableaux de type Texte, Alpha, Numérique, Date, Pointeur et Booléen. Les paramètres tableau et valeur doivent être du même type.

Si aucun élément n'est trouvé, Chercher dans tableau renvoie –1.

Si début est spécifié, Chercher dans tableau commence la recherche à l'élément spécifié par début. Si début n'est pas spécifié, Chercher dans tableau commence la recherche à l'élément 1.

Exemples

(1) La méthode projet suivante efface tous les éléments vides du tableau alpha ou texte passé en paramètre :

```
      ` Méthode projet NETTOYER TABLEAU
      ` NETTOYER TABLEAU ( Pointeur )
      ` NETTOYER TABLEAU ( -> Tableau Texte ou Alpha )
C_POINTEUR ($1)
Repeter
   $vlElem:=Chercher dans tableau ($1->;"")
   Si ($vlElem>0)
      SUPPRIMER DANS TABLEAU($1->;$vlElem)
   Fin de si
                        Jusque ($vlElem<0)
```

Une fois que cette méthode projet est implémentée dans votre base, vous pouvez écrire, par exemple :

```
TABLEAU TEXTE (TabValeurs;...)
   ` ...
   ` Utiliser le tableau comme vous voulez
   ` ...
   ` Eliminer les éléments chaînes vides
NETTOYER TABLEAU (->TabValeurs)
```

(2) La méthode projet suivante sélectionne le premier élément d'un tableau dont le pointeur passé comme premier paramètre correspond à la valeur de la variable ou du champ dont le pointeur est passé en second paramètre :

```
   ` Méthode projet SELECTIONNER ELEMENT
   ` SELECTIONNER ELEMENT ( Pointeur ; Pointeur)
   ` SELECTIONNER ELEMENT ( -> Tableau Texte ou Alpha ; -> Champ ou variable de type
                                                              Texte ou Alpha )

$1->:=Chercher dans tableau ($1->;$2->)
Si ($1->=-1)
   $1->:=0 ` Si aucun élément n'est trouvé, fixer le tableau à aucun élément sélectionné
Fin de si
```

Une fois que cette méthode projet est implémentée dans la base, vous pouvez écrire, par exemple :

```
   ` Méthode objet du pop-up menu TabTitres
Au cas ou
   : (Evenement formulaire=Sur chargement)
      SELECTIONNER ELEMENT (->TabTitres;->[Personnes]Titre)
Fin de cas
```

Référence

INSERER DANS TABLEAU, SUPPRIMER DANS TABLEAU, Taille tableau.

Compter dans tableau

Compter dans tableau (tableau; valeur) → Entier long

Paramètre	Type		Description
tableau	Tableau	→	Tableau dans lequel effectuer le comptage
valeur	Expression	→	Valeur à compter
Résultat	Entier long	←	Nombre d'occurrences trouvées

Description

La commande Compter dans tableau retourne le nombre d'occurrences de valeur dans tableau.

Cette commande peut être utilisée avec des tableaux de type Texte, Alpha, Numérique, Date, Pointeur et Booléen. Les paramètres tableau et valeur doivent être du même type ou d'un type compatible.

Si aucun élément de tableau ne correspond à valeur, la commande retourne 0.

Exemple

L'exemple suivant permet d'afficher le nombre de lignes sélectionnées dans une list box :

```
      `tBList est le nom d'un tableau de colonne List box
ALERTE(Chaine(Compter dans tableau(tBList;Vrai))+" ligne(s) sélectionnée(s) dans la list box")
```

Référence

Chercher dans tableau.

COPIER TABLEAU

COPIER TABLEAU (source; destination)

Paramètre	Type		Description
source	Tableau	→	Tableau à recopier
destination	Tableau	←	Tableau de destination

Description

La commande COPIER TABLEAU crée ou remplace le tableau destination avec les mêmes contenu, taille et type que le tableau source.

Les tableaux source et destination peuvent être des tableaux locaux, process ou interprocess. La portée du tableau n'a pas d'importance lors de la duplication des tableaux.

Exemple

L'exemple suivant remplit un tableau C. Un nouveau tableau, "D", est ensuite créé, contenant les mêmes informations que le tableau C :

```
TOUT SELECTIONNER ([Contacts]) ` Sélectionner tous les enregistrements dans la table
    ` Remplir le tableau C avec les données du champ
SELECTION VERS TABLEAU ([Contacts]Société; C)
COPIER TABLEAU (C; D) ` Copier le tableau C dans le tableau D
```

Note de compatibilité
En raison de la nouvelle implémentation des listes hiérarchiques, la compatibilité de cette fonction n'a pu être totalement maintenue. Aussi, à compter de la version 6, il est préférable d'utiliser la fonction Charger liste pour travailler avec des listes hiérarchiques définies dans l'éditeur d'énumérations, en mode Développement.

ENUMERATION VERS TABLEAU (énumération; tableau{; réfEléments})

Paramètre	Type		Description
énumération	Alpha	→	Enumération de laquelle copier les éléments du premier niveau
tableau	Tableau	←	Tableau dans lequel copier les éléments de l'énumération
réfEléments	Tableau Num	←	Numéros de référence des éléments de l'énumération

Description

La commande ENUMERATION VERS TABLEAU crée ou remplace le tableau tableau avec les éléments du premier niveau de l'énumération énumération.

Si vous n'avez pas préalablement défini le tableau comme tableau de type Alpha ou Texte, ENUMERATION VERS TABLEAU crée un tableau de type Texte par défaut.

Le paramètre optionnel réfEléments (un tableau de type Numérique) retourne les numéros de référence des éléments de l'énumération.

Note de compatibilité : Dans la version précédente de 4D, le troisième paramètre était un tableau rempli avec les noms de toutes les énumérations liées. Si un élément de l'énumération principale comportait une énumération liée, le nom de cette dernière était placée dans l'élément de tableau ayant le même numéro que l'élément de l'énumération. S'il n'y avait pas d'énumération liée, l'élément inséré était une chaîne vide. Ce second tableau avait la même taille que tableau. Vous pouviez utiliser les noms dans le tableau liens pour accéder aux énumérations liées.

Vous pouvez continuer à utiliser ENUMERATION VERS TABLEAU pour construire un tableau basé sur les éléments de premier niveau d'une énumération. Cependant, cette commande ne vous donne pas les moyens de travailler avec les éléments des sous-énumérations. Pour exploiter pleinement les listes hiérarchiques, utilisez les nouvelles commandes de listes hiérarchiques introduites par la version 6.

Exemple

L'exemple suivant recopie les éléments de l'énumération Régions dans le tableau tabRégions :

```
ENUMERATION VERS TABLEAU ("Régions"; tabRégions)
```

Référence

Charger liste, STOCKER LISTE, TABLEAU VERS ENUMERATION.

INSERER DANS TABLEAU (tableau; positionDépart{; combien})

Paramètre	Type		Description
tableau	Tableau	→	Nom du tableau dans lequel insérer des éléments
positionDépart	Numérique	→	Position de départ du ou des élément(s) à insérer
combien	Numérique	→	Nombre d'éléments à insérer ou 1 élément si ce paramètre est omis

Description

INSERER DANS TABLEAU insère un ou plusieurs éléments ou "lignes" dans le tableau tableau. Les nouveaux éléments sont insérés avant l'élément spécifié par positionDépart, et initialisés à la valeur vide du type du tableau. Tous les éléments situés au-delà de positionDépart sont décalés vers le bas d'un offset ou de la valeur spécifiée par combien.

Si positionDépartn est supérieur à la taille du tableau, les éléments sont insérés à la fin du tableau.

Le paramètre combien représente le nombre de lignes à insérer. Si combien n'est pas spécifié, un seul élément est inséré. La taille du tableau est augmentée de combien.

Exemples

(1) L'exemple suivant insère cinq nouveaux éléments à partir de l'élément 10 :

```
INSERER DANS TABLEAU (unTableau; 10; 5)
```

(2) L'exemple suivant ajoute un élément à un tableau :

```
$vlElem:=Taille tableau (unTableau)+1
INSERER DANS TABLEAU(unTableau;$vlElem)
unTableau{$vlElem}:=...
```

Référence

SUPPRIMER DANS TABLEAU, Taille tableau.

SELECTION LIMITEE VERS TABLEAU

version 5.5.3

SELECTION LIMITEE VERS TABLEAU (début; fin; leChamp|laTable; tableau{; leChamp|laTable2; tableau2; ...; leChamp|laTableN; tableauN})

Paramètre	Type		Description
début	Numérique	→	Numéro de l'enregistrement sous-sélectionné à partir duquel commencer la copie des données
fin	Numérique	→	Numéro de l'enregistrement sous-sélectionné auquel arrêter la copie des données
leChamp|laTable	Champ ou Table	→	Champ à utiliser pour récupérer les données ou Table à utiliser pour récupérer les numéros d'enregistrements
tableau	Tableau	←	Tableau recevant les données ou les numéros d'enregistrements

Description

SELECTION LIMITEE VERS TABLEAU crée un ou plusieurs tableaux et y copie des données en provenance des champs de la sélection courante ou les numéros des enregistrements de la sélection courante.

A la différence de SELECTION VERS TABLEAU qui s'applique à l'intégralité de la sélection courante, SELECTION LIMITEE VERS TABLEAU s'applique uniquement à une sous-sélection d'enregistrements, définie par les paramètres début et fin.

Vous devez passer dans les paramètres début et fin des numéros d'enregistrements sous-sélectionnés s'inscrivant dans l'intervalle défini par la formule 1 <= début <= fin <= Enregistrements trouves ([...]).

Si vous passez des numéros correspondant à 1 <= début = fin <= Enregistrements trouves ([...]), ce sont les champs ou les numéros des enregistrements de la sélection courante répondant à début = fin qui seront chargés.

Si vous passez des numéros d'enregistrements incorrects, vous obtiendrez les résultats suivants :
• Si fin > Enregistrements trouves ([...]), la commande retourne toutes les valeurs, à partir de l'enregistrement sous-sélectionné spécifié par début jusqu'au dernier enregistrement sous-sélectionné.

• Si début > fin, la commande ne retourne que les valeurs de l'enregistrement début.
• Si les deux paramètres sont incompatibles avec la taille de la sous-sélection, les tableaux sont retournés vides

Comme SELECTION VERS TABLEAU, SELECTION LIMITEE VERS TABLEAU s'applique à la sélection de la table passée en paramètre. La commande peut réaliser les opérations suivantes :
• Charger les valeurs d'un ou plusieurs champs,
• Charger les numéros des enregistrements, à l'aide de la syntaxe ...;[table];tableau;...
• Charger des valeurs de champs liés, s'il existe un lien automatique de N vers 1 entre les tables, ou si vous avez préalablement appelé la commande FIXER LIENS AUTOMATIQUES pour rendre automatiques les liens manuels N vers 1 (dans les deux cas, les valeurs peuvent être chargées à travers plusieurs niveaux de liens N vers 1 entre les tables).

Chaque tableau est typé en fonction du type de champ. Il y a cependant deux exceptions :
• La copie d'un champ de type Heure provoquera la création d'un tableau Entier long.
• En mode compatibilité ASCII (non Unicode), lorsqu'un champ de type Texte est copié dans un tableau Alpha, le tableau reste de type Alpha.

Si vous chargez les numéros des enregistrements, ils sont copiés dans un tableau de type Entier long.

4D Server : La commande SELECTION LIMITEE VERS TABLEAU est optimisée pour 4D Server. Chaque tableau est créé sur le serveur puis envoyé en totalité sur le poste client.

ATTENTION : SELECTION LIMITEE VERS TABLEAU peut créer des tableaux de taille importante, en fonction de l'intervalle défini par début et fin, ainsi que du type et de la taille des données à charger. Comme les tableaux résident en mémoire, il peut être utile de tester la taille des tableaux créés après l'exécution de la commande, ou d'utiliser une méthode projet d'interception d'erreurs, installée par la commande APPELER SUR ERREUR.

Une fois la commande correctement exécutée, la taille des tableaux résultants est égale à (fin-début)+1 — sauf si le paramètre fin est supérieur au nombre d'enregistrements dans la sélection. Dans ce cas, les tableaux contiennent (Enregistrements trouves([...])-début)+1 éléments.

Exemples

(1) La ligne de code suivante utilise les 50 premiers enregistrements de la sélection courante de la table [Factures]. Les valeurs du champ [Factures]RéfFacture et du champ lié [Clients]RéfClient sont chargées.

```
SELECTION LIMITEE VERS TABLEAU (1;50;[Factures]RéfFacture;tlRéfFacture;
[Clients]RéfClient;tlRéfClient)
```

(2) Les lignes de code suivantes utilisent les 50 derniers enregistrements de la sélection courante de la table [Factures]. Les numéros d'enregistrements de la table [Factures] ainsi que ceux de la table liée [Clients] sont chargés :

```
lTailleSél := Enregistrements trouves ([Factures])
SELECTION LIMITEE VERS TABLEAU (lTailleSél-49;lTailleSél;[Factures];taFactureNum;
                                                              [Clients];taClientNum)
```

(3) Les lignes de code suivantes vous permettent de travailler séquentiellement avec des portions de 1000 enregistrements d'une sélection importante qui ne peut pas être chargée dans des tableaux en une seule fois :

```
lMaxPage := 1000
lTailleSél := Enregistrements trouves ([Annuaire])
Boucle ($lPage ; 1; 1+((lTailleSél-1)\lMaxPage) )
      ` Charger les valeurs et/ou les numéros d'enregistrements
   SELECTION LIMITEE VERS TABLEAU (1+(lMaxPage*($lPage-1));lMaxPage*$lPage
                                                          ;...;...;...;...;...;...)
      ` Faire quelque chose avec les tableaux
Fin de boucle
```

Référence

APPELER SUR ERREUR, FIXER LIENS AUTOMATIQUES, SELECTION VERS TABLEAU.

SELECTION VERS TABLEAU

SELECTION VERS TABLEAU (leChamp|laTable; tableau{; leChamp|laTable2; tableau2; ...; leChamp|laTableN; tableauN})

Paramètre	Type		Description
leChamp|laTable	Champ ou Table	→	Champ à récupérer dans le tableau ou Table dont les numéros d'enregistrements sont à récupérer dans le tableau
tableau	Tableau	←	Tableau recevant les valeurs des champs ou les numéros d'enregistrements

Description

La commande SELECTION VERS TABLEAU crée un ou plusieurs tableaux et y copie les valeurs des champ(s) ou les numéros d'enregistrement(s) de la sélection courante.

SELECTION VERS TABLEAU s'applique à la sélection courante de la table spécifiée dans le premier paramètre. La commande peut réaliser les opérations suivantes :
• Charger les valeurs d'un ou plusieurs champs,
• Charger les numéros des enregistrements, à l'aide de la syntaxe ...;[table];tableau;...
• Charger des valeurs de champs liés, s'il existe un lien automatique de N vers 1 entre les tables, ou si vous avez préalablement appelé la commande FIXER LIENS AUTOMATIQUES pour rendre automatiques les liens manuels N vers 1 (dans les deux cas, les valeurs peuvent être chargées à travers plusieurs niveaux de liens N vers 1 entre les tables).

Chaque tableau est typé en fonction du type de champ. Il y a cependant deux exceptions :
• La copie d'un champ de type Heure provoquera la création d'un tableau Entier long.
• En mode compatibilité ASCII (non Unicode), lorsqu'un champ de type Texte est copié dans un tableau Alpha, le tableau reste de type Alpha.

Si vous chargez les numéros des enregistrements, ils sont copiés dans un tableau de type Entier long.

4D Server : La commande SELECTION VERS TABLEAU est optimisée pour 4D Server. Chaque tableau est créé sur le serveur puis envoyé en totalité sur le poste client.

ATTENTION : SELECTION VERS TABLEAU peut créer des tableaux de taille importante, en fonction de la taille de la sélection courante, ainsi que du type et de la taille des données à charger. Comme les tableaux résident en mémoire, il peut être utile de tester la taille des tableaux créés après l'exécution de la commande, ou d'utiliser une méthode projet d'appel sur erreur.

Note : Après un appel à SELECTION VERS TABLEAU, la sélection courante et l'enregistrement courant ne sont pas modifiés, mais l'enregistrement courant n'est plus chargé. Utilisez la commande CHARGER ENREGISTREMENT après un SELECTION VERS TABLEAU si vous souhaitez utiliser les valeurs des champs de l'enregistrement courant.

Exemples

(1) Dans l'exemple suivant, la table [Personnes] dispose d'un lien automatique vers la table [Sociétés]. Les deux tableaux tabNoms et tabAdresseSociétés sont dimensionnés en fonction du nombre d'enregistrements dans la sélection de la table [Personnes] et contiennent des informations venant des deux tables :

```
SELECTION VERS TABLEAU ([Personnes]Nom; tabNoms; [Sociétés]Adresse;
                                                    tabAdresseSociétés)
```

(2) L'exemple ci-dessous retourne les numéros d'enregistrements de la table [Clients] dans le tableau tabNumEnr et les valeurs du champ [Clients]Noms dans le tableau tabNoms :

```
SELECTION VERS TABLEAU([Clients]; tabNumEnr; [Clients]Noms; tabNoms)
```

Référence

APPELER SUR ERREUR, FIXER LIENS AUTOMATIQUES, SELECTION LIMITEE VERS TABLEAU, TABLEAU MULTI TRI, TABLEAU VERS SELECTION.

SUPPRIMER DANS TABLEAU

version 11 (Modifiée)

SUPPRIMER DANS TABLEAU (tableau; positionDépart{; combien})

Paramètre	Type		Description
tableau	Tableau	→	Tableau dans lequel supprimer des lignes
positionDépart	Numérique	→	Elément de départ de la suppression
combien	Numérique	→	Nombre d'éléments à supprimer ou 1 élément si ce paramètre est omis

Description

La commande SUPPRIMER DANS TABLEAU supprime un ou plusieurs élément(s) de tableau. Les éléments sont supprimés à partir de l'élément spécifié par positionDépart.

Le paramètre combien est le nombre d'éléments à supprimer. Si combien n'est pas spécifié, un seul élément est supprimé. La taille du tableau est réduite de combien.

Exemples

(1) L'exemple suivant supprime trois éléments en commençant à l'élément 5 :

```
SUPPRIMER DANS TABLEAU(unTableau; 5; 3)
```

(2) L'exemple suivant supprime le dernier élément d'un tableau, s'il existe :

```
$vlElem:=Taille tableau (unTableau)
Si ($vlElem>0)
   SUPPRIMER DANS TABLEAU (unTableau;$vlElem)
Fin de si
```

Référence

INSERER DANS TABLEAU, Taille tableau.

TABLEAU ALPHA (longueurChaîne; nomTableau; taille{; taille2})

Paramètre	Type		Description
longueurChaîne	Numérique	→	Longueur de la chaîne (1..255)
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU ALPHA crée et/ou redimensionne un tableau d'éléments de type Chaîne alphanumérique en mémoire.

Note de compatibilité : Les bases de données créées à compter de la version 11 de 4D sont exécutées par défaut en mode Unicode (cf. section Codes ASCII). Dans ce mode, le fonctionnement de la commande TABLEAU ALPHA est rigoureusement identique à la celui de commande TABLEAU TEXTE (le paramètre longueurChaîne est ignoré). Il est conseillé d'utiliser TABLEAU TEXTE dans les nouveaux développements. La commande TABLEAU ALPHA est conservée pour des raisons de compatibilité uniquement.

• Le paramètre longueurChaîne définit le nombre maximum de caractères que chaque élément du tableau peut contenir. Cette longueur doit être comprise entre 1 et 255 caractères.
Note : Ce paramètre est pris en compte lorsque la base est éxécutée en mode non Unicode uniquement. En mode Unicode, il est ignoré (voir *Note de compatibilité* ci-dessus).
• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU ALPHA à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à "" (chaîne vide).
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process comportant 100 éléments. Chaque élément est une chaîne de 31 caractères :

```
TABLEAU ALPHA (31;tabAlpha;100)
```

(2) Cet exemple crée un tableau local de 100 lignes. Chaque ligne comporte 50 éléments, chaque élément est une chaîne de 63 caractères :

```
TABLEAU ALPHA (63;$tabAlpha;100;50)
```

(3) Cet exemple crée un tableau interprocess comportant 50 éléments. Chaque élément est une chaîne de 255 caractères. La valeur "Elément No" suivie du numéro de l'élément est affectée à chaque élément :

```
TABLEAU ALPHA (255;<>tabAlpha;50)
Boucle ($vlElem;1;50)
    <>tabAlpha{$vlElem}:="Elément No" + Chaine($vlElem)
Fin de boucle
```

Référence

TABLEAU TEXTE.

TABLEAU BOOLEEN

TABLEAU BOOLEEN (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU BOOLEEN crée et/ou redimensionne un tableau d'éléments de type Booléen en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU BOOLEEN à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à Faux.
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Astuce : Dans certaines circonstances, l'utilisation d'un tableau d'Entiers dans lequel chaque élément différent de zéro signifie "vrai" et chaque élément égal à zéro signifie "faux" est une alternative à l'utilisation d'un tableau de Booléens.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Booléen :

```
TABLEAU BOOLEEN (tabBooléens; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Booléen :

```
TABLEAU BOOLEEN ($tabBooléens;100;50)
```

(3) Cet exemple crée un tableau interprocess de 50 éléments de type Booléen et affecte à chaque élément pair la valeur Faux :

```
TABLEAU BOOLEEN (<>tabBooléens;50)
Boucle($vElem;1;50)
    <>tabBooléens{$vElem}:=(($vElem%2)=0)
Fin de boucle
```

Référence

TABLEAU ENTIER.

TABLEAU BOOLEEN SUR ENSEMBLE

version 6.5

TABLEAU BOOLEEN SUR ENSEMBLE (tabBooléen{; ensemble})

Paramètre	Type		Description
tabBooléen	Tab booléen	←	Tableau d'appartenance des enregistrements à l'ensemble
ensemble	Alpha	→	Nom de l'ensemble ou Ensemble UserSet si ce paramètre est omis

Description

La commande TABLEAU BOOLEEN SUR ENSEMBLE remplit un tableau de booléens indiquant si chaque enregistrement de la table à laquelle appartient ensemble fait ou non partie de l'ensemble.

Les éléments du tableau sont ordonnés en fonction de l'ordre de création des enregistrements dans la table (numéros absolus). Si N est le nombre d'enregistrements de la table, l'élément 0 du tableau correspond à l'enregistrement n° 0, l'élément 1 du tableau correspond à l'enregistrement n° 1, etc.

Chaque élément du tableau est mis à :

- Vrai si l'enregistrement correspondant fait partie de l'ensemble,
- Faux si l'enregistrement correspondant ne pas partie de l'ensemble.

Attention, le nombre total d'éléments du tableau tabBooléen n'est pas significatif. En effet, pour des raisons structurelles, il peut être différent du nombre d'enregistrements effectivement présents dans la table. Les éventuels éléments supplémentaires sont mis à Faux.

Si vous ne passez pas le paramètre ensemble, la commande utilisera l'ensemble système *UserSet* du process courant.

Référence

CREER ENSEMBLE SUR TABLEAU.

TABLEAU DATE (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU DATE crée et/ou redimensionne un tableau d'éléments de type Date en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU DATE à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à la valeur de date nulle (!00/00/00!).
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Date :

```
TABLEAU DATE (tabDates; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Date :

```
TABLEAU DATE ($tabDates;100;50)
```

(3) Cet exemple crée un tableau interprocess de 50 éléments de type Date et affecte à chaque élément la date du jour + un nombre de jours égal au numéro de l'élément :

```
TABLEAU DATE (<>tabDates;50)
Boucle($vElem;1;50)
    <>tabDates{$vElem}:=Date du jour+$vElem
Fin de boucle
```

TABLEAU ENTIER (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU ENTIER crée et/ou redimensionne un tableau d'éléments de type Entier (2 octets) en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU ENTIER à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à 0.
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Entier :

```
TABLEAU ENTIER (tabEntiers; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Entier :

```
TABLEAU ENTIER ($tabEntiers;100;50)
```

(3) Cet exemple crée un tableau interprocess de 50 éléments de type Entier et affecte à chaque élément son numéro :

```
TABLEAU ENTIER (<>tabEntiers; 50)
Boucle($vElem;1;50)
    <>tabEntiers{$vElem}:=$vElem
Fin de boucle
```

Référence

TABLEAU ENTIER LONG, TABLEAU REEL.

TABLEAU ENTIER LONG

version 5

TABLEAU ENTIER LONG (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU ENTIER LONG crée et/ou redimensionne un tableau d'éléments de type Entier long (4 octets) en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU ENTIER LONG à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à 0.
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Entier long :

```
TABLEAU ENTIER LONG (tabEntiersLongs; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Entier long :

```
TABLEAU ENTIER LONG ($tabEntiersLongs;100;50)
```

(3) Cet exemple crée un tableau interprocess de 50 éléments de type Entier long et affecte à chaque élément son numéro :

```
TABLEAU ENTIER LONG (<>tabEntiersLongs; 50)
Boucle($vElem;1;50)
    <>tabEntiersLongs{$vElem}:=$vElem
Fin de boucle
```

Référence

TABLEAU ENTIER, TABLEAU REEL.

TABLEAU ENTIER LONG SUR SELECTION

version 6.7 (Modifiée)

TABLEAU ENTIER LONG SUR SELECTION (table; tabEnrg{; tempo})

Paramètre	Type		Description
table	Table	→	Table de la sélection courante
tabEnrg	Tab Entier long	←	Tableau de numéros d'enregistrements
tempo	Alpha	→	Nom de la sélection temporaire ou Sélection courante si ce paramètre est omis

Description

La commande TABLEAU ENTIER LONG SUR SELECTION remplit le tableau tabEnrg avec les numéros (absolus) des enregistrements faisant partie de la sélection temporaire tempo.

Si vous ne passez pas le paramètre tempo, la commande utilise la sélection courante de la table table.

Note : L'élément n° 0 du tableau tabEnrg est initialisé à -1.

Référence

CREER SELECTION SUR TABLEAU.

TABLEAU IMAGE

TABLEAU IMAGE (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU IMAGE crée et/ou redimensionne un tableau d'éléments de type Image en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU IMAGE à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à une image vide (ce qui signifie que la fonction Taille image appliquée à l'un de ces éléments retourne 0).
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Image :

```
TABLEAU IMAGE (tabImages; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Image :

```
TABLEAU IMAGE ($tabImages;100;50)
```

(3) Cet exemple crée un tableau interprocess d'éléments de type Image. La taille du tableau est égale au nombre de ressources 'PICT' dont le nom commence par "Utilisateur Intf/" disponibles dans la base. Chaque image est chargée dans un élément du tableau :

```
LISTE RESSOURCES ("PICT";$aiResIDs;$asResNoms)
TABLEAU IMAGE (<>tabImages;Taille tableau($aiResIDs))
$vlPictElem:=0
Boucle ($vlElem;1;Taille tableau(<>tabImages))
    Si ($asResNoms{$vlElem}="Utilisateur Intf/@")
        $vlPictElem:=$vlPictElem+1
        LIRE RESSOURCE IMAGE("PICT";$aiResIDs{$vlElem};$vgImage)
        <>tabImages{$vlPictElem}:=$vgImage
    Fin de si
Fin de boucle
TABLEAU IMAGE (<>tabImages;$vlPictElem)
```

TABLEAU MULTI TRI (tableau{; sensDuTri}{; tableau2; sensDuTri2; ...; tableauN; sensDuTriN})

Paramètre	Type		Description
tableau	Tableau	→	Tableau(x) à trier
sensDuTri	> ou <	→	> pour effectuer un tri croissant ou < pour effectuer un tri décroissant Si omis = pas de tri

TABLEAU MULTI TRI (nomTabPtr; nomTabTri)

Paramètre	Type		Description
nomTabPtr	Tableau pointeur	→	Tableau de pointeurs de tableaux
nomTabTri	Tableau Entier long	→	Tableau d'ordres de tri (1 = tri par ordre croissant, -1 = tri par ordre décroissant, 0 = synchronisation avec des tris précédents)

Description

La commande TABLEAU MULTI TRI vous permet d'effectuer un tri multi-critères sur un ensemble de tableaux. Cette fonction est notamment utile dans le cadre des zones de défilement groupées dans les formulaires.

Cette commande admet deux syntaxes différentes.

• **Première syntaxe :** TABLEAU MULTI TRI **(tableau{; sensDuTri}{; tableau2; sensDuTri2; ...; tableauN; sensDuTriN})**
Cette syntaxe est la plus simple, elle permet de passer directement les noms des tableaux synchronisés auxquels vous souhaitez appliquer un tri multi-critères.

Vous pouvez passer un nombre illimité de couples (tableau;> ou <) et/ou de tableaux seuls. Tous les tableaux passés en paramètres sont triés de manière synchronisée.

Vous pouvez passer des tableaux de tout type, à l'exception des tableaux de pointeurs et d'images. Vous pouvez trier un élément d'un tableau à deux dimensions (c'est-à-dire t2DTableau{$vlCetElément}), mais vous ne pouvez pas trier le tableau 2D lui-même (c'est-à-dire t2DTableau).

Pour utiliser le contenu d'un tableau comme critère de tri, passez le paramètre sensDuTri. La valeur du paramètre (> ou <) définit l'ordre (croissant ou décroissant) dans lequel le tableau sera trié. Si le paramètre sensDuTri est omis, le contenu du tableau n'est pas utilisé comme critère de tri.

Note : Attention, au moins un critère de tri doit être passé pour que la commande fonctionne. Si aucun critère de tri n'est défini, une erreur est générée.

Les niveaux de tris sont déterminés par l'ordre dans lequel les tableaux sont passés à la commande : la position d'un tableau avec critère dans la syntaxe détermine son niveau de tri.

• Seconde syntaxe : TABLEAU MULTI TRI (nomTabPtr; nomTabTri)
Cette syntaxe, plus complexe, est précieuse pour les développements génériques (par exemple, vous pouvez créer une méthode générique de tri des tableaux de tout type, ou encore générer l'équivalent d'un TRIER TABLEAU générique).

La paramètre nomTabPtr contient le nom d'un tableau de pointeurs de tableaux ; chaque élément de ce tableau est un pointeur désignant un tableau à trier. Les tris seront effectués dans l'ordre des pointeurs de tableaux défini par nomTabPtr. **Attention**, tous les tableaux pointés par nomTabPtr doivent avoir le même nombre d'éléments.

Note : nomTabPtr peut être un tableau de pointeurs local (*$nomTabPtr*), process (*nomTabPtr*) ou interprocess (*<>nomTabPtr*). En revanche, les éléments de ce tableau doivent pointer sur des tableaux process ou interprocess uniquement.

La paramètre nomTabTri contient le nom d'un tableau dont chaque élément indique l'ordre de tri (-1, 0 ou 1) de l'élément du tableau de pointeurs correspondant :
-1 = Tri par ordre décroissant.
0 = Le tableau n'est pas utilisé comme critère de tri mais doit être trié en fonction des autres tris.
1 = Tri par ordre croissant.

Note : Vous ne pouvez pas trier de tableaux de type Pointeur ou Image. Vous pouvez trier un élément d'un tableau à deux dimensions (c'est-à-dire t2DTableau{$vlCetElément}), mais vous ne pouvez pas trier le tableau 2D lui-même (c'est-à-dire t2DTableau).

A chaque élément du tableau nomTabPtr doit correspondre un élément du tableau nomTabTri. Les deux tableaux doivent donc avoir exactement le même nombre d'éléments.

Exemples

(1) L'exemple suivant utilise la première syntaxe : il crée quatre tableaux et les trie par ville (ordre croissant) puis par salaire (ordre décroissant), les deux derniers tableaux tab_Noms et tab_NumTel étant synchronisés en fonction des critères de tri précédents :

```
TOUT SELECTIONNER([Employés])
SELECTION VERS TABLEAU([Employés]Ville;tab_Villes;[Employés]Salaire;tab_Salaire;
                       [Employés]Nom;tab_Noms;[Employés]NumTel;tab_NumTel)
TABLEAU MULTI TRI (tab_Villes;>;tab_Salaire;<;tab_Noms;tab_NumTel)
```

Si vous souhaitez que le tableau des noms soit utilisé comme troisième critère de tri, il vous suffit d'ajouter > ou < derrière le paramètre tab_Noms.
A noter que la syntaxe :

```
TABLEAU MULTI TRI (tab_Villes;>;tab_Salaire;tab_Noms;tab_NumTel)
```

équivaut strictement à :

```
TRIER TABLEAU(tab_Villes;tab_Salaire;tab_Noms;tab_NumTel;>)
```

(2) L'exemple suivant utilise la seconde syntaxe : il crée quatre tableaux et les trie par ville (ordre croissant) et société (ordre décroissant), les deux derniers tableaux *tab_Noms* et *tab_NumTel* étant synchronisés en fonction des critères de tri précédents :

```
TOUT SELECTIONNER([Employés])
SELECTION VERS TABLEAU([Employés]Ville;tab_Villes;[Employés]Société;tab_Société;
                       [Employés]Nom;tab_Noms;[Employés]NumTel;tab_NumTel)
TABLEAU POINTEUR(tab_Pointeurs;4)
TABLEAU ENTIER LONG(tab_Tris;4)
tab_Pointeurs{1}:=->tab_Villes
tab_Tris{1}:=1
tab_Pointeurs{2}:=->tab_Société
tab_Tris{2}:=-1
tab_Pointeurs{3}:=->tab_Noms
tab_Tris{3}:=0
tab_Pointeurs{4}:=->tab_NumTel
tab_Tris{4}:=0
TABLEAU MULTI TRI (tab_Pointeurs;tab_Tris)
```

Si vous souhaitez que le tableau des noms soit utilisé comme troisième critère de tri, il vous suffit d'assigner la valeur 1 à l'élément tab_Tris{3}. Ou bien, si vous souhaitez que les tableaux soient triés uniquement sur le critère des villes, assignez la valeur 0 aux éléments tab_Tris{2}, tab_Tris{3} et tab_Tris{4}. De cette manière, vous obtenez un résultat identique à TRIER TABLEAU(tab_Villes;tab_Société;tab_Noms;tab_NumTel;>).

Référence

SELECTION VERS TABLEAU, TRIER, TRIER TABLEAU.

TABLEAU POINTEUR (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU POINTEUR crée ou redimensionne un tableau d'éléments de type Pointeur en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU POINTEUR à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à un pointeur nul (ce qui signifie que la fonction Nil appliquée à l'un de ces éléments retourne Vrai).
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Pointeur :

```
TABLEAU POINTEUR (tabPointeurs; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Pointeur :

```
TABLEAU POINTEUR ($tabPointeurs;100;50)
```

(3) Cet exemple crée un tableau interprocess d'éléments de type Pointeur dont la taille est égale au nombre de tables dans la base et remplit chaque élément pointant vers la table dont le numéro est le même que celui de l'élément. Dans la cas d'une table supprimée, la ligne retournera Nil.

```
TABLEAU POINTEUR (<>tabPointeurs;Lire numero derniere table)
Boucle($vElem;Taille tableau(<>tabPointeurs);1;-1)
   Si(Est un numero de table valide($vElem))
      <>tabPointeurs{$vElem}:=Table($vElem)
Fin de boucle
```

TABLEAU REEL (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU REEL crée et/ou redimensionne un tableau d'éléments de type Réel (numérique) en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU REEL à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à 0.
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Réel :

```
TABLEAU REEL (tabRéel; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Réel :

```
TABLEAU REEL ($tabRéel; 100;50)
```

(3) Cet exemple crée un tableau interprocess de 50 éléments de type Réel et affecte à chaque élément son numéro :

```
TABLEAU REEL (<>tabRéel; 50)
Boucle($vElem;1;50)
    <>tabRéel{$vElem}:=$vElem
Fin de boucle
```

Référence

TABLEAU ENTIER, TABLEAU ENTIER LONG.

TABLEAU TEXTE (nomTableau; taille{; taille2})

Paramètre	Type		Description
nomTableau	Tableau	→	Nom du tableau
taille	Numérique	→	Nombre d'éléments du tableau ou Nombre de tableaux si taille2 est spécifié
taille2	Numérique	→	Nombre d'éléments des tableaux à deux dimensions

Description

La commande TABLEAU TEXTE crée et/ou redimensionne un tableau d'éléments de type Texte en mémoire.

• Le paramètre nomTableau est le nom du tableau.
• Le paramètre taille est le nombre d'éléments du tableau.
• Le paramètre taille2 est optionnel. Si vous le spécifiez, cette commande crée un tableau à deux dimensions. Dans ce cas, taille spécifie le nombre de lignes et taille2 spécifie le nombre de colonnes de chaque tableau. Chaque ligne dans un tableau à deux dimensions peut être traitée à la fois comme un élément et comme un tableau. Cela signifie que vous pouvez insérer et supprimer des tableaux entiers dans un tableau à deux dimensions, par l'intermédiaire des autres commandes de ce thème, lorsque vous travaillez avec la première dimension du tableau.

Lorsque vous appliquez la commande TABLEAU TEXTE à un tableau existant :
• Si vous agrandissez sa taille, les éléments existants ne sont pas modifiés, les nouveaux éléments sont initialisés à "" (chaîne vide).
• Si vous réduisez sa taille, les éléments du "bas" du tableau sont supprimés et perdus.

Exemples

(1) Cet exemple crée un tableau process contenant 100 éléments de type Texte :

```
TABLEAU TEXTE (tabTexte; 100)
```

(2) Cet exemple crée un tableau local de 100 lignes contenant chacune 50 éléments de type Texte :

```
TABLEAU TEXTE ($tabEntiersLongs;100;50)
```

(3) Cet exemple crée un tableau interprocess de 50 éléments de type Texte et affecte à chaque élément la valeur "Elément No" suivie du numéro de l'élément :

```
TABLEAU TEXTE (<>tabTexte; 50)
Boucle($vElem;1;50)
    <>tabTexte{$vElem}:="Elément n°"+ Chaine ($vElem)
Fin de boucle
```

Référence

TABLEAU ALPHA.

TABLEAU VERS ENUMERATION

version 5

Note de compatibilité
En raison de la nouvelle implémentation des listes hiérarchiques, la compatibilité de cette commande n'a pu être totalement maintenue. Aussi, à compter de la version 6, il est préférable d'utiliser la commande STOCKER LISTE pour travailler avec des listes hiérarchiques définies dans l'éditeur d'énumérations, en mode Développement.

TABLEAU VERS ENUMERATION (tableau; énumération{; réfEléments})

Paramètre	Type		Description
tableau	Tableau	→	Tableau duquel copier les éléments
énumération	Alpha	←	Enumération dans laquelle copier les éléments de tableau
réfEléments	Tableau Num	→	Tableau numérique des numéros de référence des éléments

Description

La commande TABLEAU VERS ENUMERATION crée ou remplace l'énumération énumération (définie dans l'éditeur d'énumérations en mode Développement) en utilisant les éléments du tableau tableau.

Cette commande vous permet de définir seulement les éléments du premier niveau de l'énumération.

Le paramètre optionnel réfEléments, s'il est passé, doit être un tableau de type Numérique synchronisé avec le tableau tableau. Chaque élément de ce tableau indique le numéro de référence de l'élément de l'énumération correspondant dans tableau. Si ce paramètre est omis, 4D affecte automatiquement aux éléments de l'énumération les numéros de référence 1, 2... N.

Note de compatibilité : Dans la version précédente de 4D, ce paramètre était utilisé pour lier d'autres énumérations à chaque élément de tableau. Si un élément du tableau liens correspondait au nom d'une énumération existante, cette énumération était alors rattachée à l'élément correspondant.

Vous pouvez continuer à utiliser TABLEAU VERS ENUMERATION pour construire une énumération basée sur les éléments d'un tableau. Cependant, cette commande ne vous donne pas les moyens de travailler avec les éléments des sous-énumérations. Pour exploiter pleinement les listes hiérarchiques, utilisez les nouvelles commandes de listes hiérarchiques introduites avec la version 6 de 4D.

Exemple

L'exemple suivant copie le tableau tabRégions dans l'énumération "Régions" :

```
TABLEAU VERS ENUMERATION (tabRégions; "Régions")
```

Référence

APPELER SUR ERREUR, Charger liste, ENUMERATION VERS TABLEAU, STOCKER LISTE.

Gestion des erreurs

La commande TABLEAU VERS ENUMERATION génère l'erreur -9957 lorsqu'elle est appliquée à une énumération en cours de modification en mode Développement. Vous pouvez intercepter cette erreur à l'aide d'une méthode projet de gestion des erreurs installée par la commande APPELER SUR ERREUR.

TABLEAU VERS SELECTION (tableau; champ{; tableau2; champ2; ...; tableauN; champN})

Paramètre	Type		Description
tableau	Tableau	→	Tableau à copier dans la sélection
champ	Champ	←	Champ recevant les valeurs du tableau

Description

La commande TABLEAU VERS SELECTION copie un ou plusieurs tableaux vers une sélection d'enregistrements. Tous les champs listés doivent appartenir à la même table.

Si une sélection existe au moment de l'appel, les éléments du tableau sont copiés dans les enregistrements en fonction de l'ordre du tableau et de l'ordre des enregistrements. Si le nombre d'éléments du tableau est supérieur au nombre d'enregistrements de la sélection courante, de nouveaux enregistrements sont créés. Les enregistrements, qu'ils soient nouveaux ou existants, sont automatiquement sauvegardés.

Tous les tableaux doivent avoir le même nombre d'éléments. Si des tableaux ont des tailles différentes, une erreur de syntaxe est générée.

Cette commande effectue l'opération inverse de SELECTION VERS TABLEAU. Cependant, TABLEAU VERS SELECTION ne permet pas d'utiliser de champs en provenance de tables différentes ni de tables liées, même si un lien automatique existe.

ATTENTION : Comme TABLEAU VERS SELECTION remplace les informations éventuellement présentes dans les enregistrements existants, cette commande doit être utilisée avec prudence. Si un enregistrement est verrouillé par un autre process pendant l'exécution de la commande TABLEAU VERS SELECTION, il n'est pas modifié. Tous les enregistrements verrouillés sont placés dans l'ensemble LockedSet. Après l'exécution de TABLEAU VERS SELECTION, vous pouvez tester si l'ensemble LockedSet contient des enregistrements qui étaient verrouillés.

4D Server : Cette commande est optimisée pour 4D Server. Le tableau est envoyé au serveur depuis le poste client. Les enregistrements sont modifiés ou créés sur le serveur. Comme une telle requête est gérée de façon synchrone, le poste client doit attendre que l'opération se soit correctement déroulée. Dans les environnements multi-utilisateurs et multi-process, aucun enregistrement verrouillé ne sera réécrit.

Exemple

Dans l'exemple suivant, les deux tableaux tabNoms et tabSociétés écrivent des données dans la table [Personnes]. Les valeurs du tableau tabNoms sont placées dans le champ [Personnes]Nom et les valeurs du tableau tabSociétés sont placées dans le champ [Personnes]Société :

```
TABLEAU VERS SELECTION (tabNoms; [Personnes]Nom; tabSociétés; [Personnes]Société)
```

Référence

SELECTION VERS TABLEAU.

Taille tableau

Taille tableau (tableau) → Numérique

Paramètre	Type		Description
tableau	Tableau	→	Tableau dont vous désirez connaître la taille
Résultat	Numérique	←	Nombre d'éléments dans le tableau

Description

Taille tableau retourne le nombre d'éléments de tableau.

Exemples

(1) L'exemple suivant retourne la taille du tableau monTableau :

```
vTaille := Taille tableau (monTableau)  ` vTaille reçoit la taille de monTableau
```

(2) L'exemple suivant retourne le nombre de lignes d'un tableau à deux dimensions :

```
vlLignes:=Taille tableau(t2DTableau)  ` vlLignes reçoit la taille de t2DTableau
```

(3) L'exemple suivant retourne le nombre de colonnes d'une ligne d'un tableau à deux dimensions :

```
vlColonnes:=Taille tableau(t2DTableau{10})  ` vlColonnes reçoit la taille de t2DTableau{10}
```

Référence

INSERER LIGNES, SUPPRIMER LIGNES.

TRIER TABLEAU

TRIER TABLEAU (tableau{; tableau2; ...; tableauN}{; sensDuTri})

Paramètre	Type		Description
tableau	Tableau	→	Tableau(x) à trier
sensDuTri	> ou <	→	> pour effectuer un tri par ordre croissant ou < pour effectuer un tri par ordre décroissant Tri croissant si ce paramètre est omis

Description

La commande TRIER TABLEAU trie un ou plusieurs tableaux par ordre croissant ou décroissant.

Note : Vous ne pouvez pas trier de tableaux de type Pointeur ou Image. Vous pouvez trier un élément d'un tableau à deux dimensions (c'est-à-dire t2DTableau{$vlCetElément}), mais vous ne pouvez pas trier le tableau 2D lui-même (c'est-à-dire t2DTableau).

Le paramètre sensDuTri spécifie l'ordre du tri : croissant ou décroissant. Si sensDuTri est égal au symbole “supérieur à” (>), l'ordre du tri est croissant. Si sensDuTri est égal au symbole “inférieur à” (<), l'ordre du tri est décroissant. Si sensDuTri est omis, l'ordre du tri est croissant.

Si plus d'un tableau est spécifié, les tableaux sont triés en fonction de l'ordre défini pour le premier tableau (les tris multi-niveaux ne sont pas possibles dans ce cas). Ce fonctionnement est particulièrement utile avec les zones de défilement dans des formulaires : TRIER TABLEAU assure la synchronisation des tableaux utilisés pour ces zones.

Exemples

(1) L'exemple suivant crée deux tableaux et les trie en fonction du nom de la société :

```
TOUT SELECTIONNER ([Personnes])
SELECTION VERS TABLEAU ([Personnes]Noms; tabNoms;[Personnes]Sociétés; tabSociétés)
TRIER TABLEAU (tabSociétés; tabNoms; >)
```

Cependant, comme TRIER TABLEAU n'effectue pas de tris multi-niveaux, les noms des personnes apparaîtront en désordre à l'intérieur de chaque société. Pour que les noms des personnes soient triés pour chaque société, vous devrez plutôt écrire :

```
TOUT SELECTIONNER ([Personnes])
TRIER ([Personnes];[Personnes]Sociétés; >;[Personnes]Noms;>)
SELECTION VERS TABLEAU ([Personnes]Noms; tabNoms;[Personnes]Sociétés; tabSociétés)
```

(2) Vous affichez les noms d'une table [Personnes] dans une fenêtre flottante. Cette liste de noms peut être triée de A vers Z ou de Z vers A en fonction du bouton sur lequel vous cliquez, dans la fenêtre. Comme il se peut que certaines personnes portent le même nom, vous avez également créé un champ [Personnes]Numéro ID qui est un champ indexé unique. Lorsque vous cliquez sur un nom dans la liste, vous voulez récupérer l'enregistrement correspondant. En utilisant un tableau synchronisé et caché des numéros d'ID, vous êtes certain d'accéder à l'enregistrement correspondant au nom sélectionné :

```
    ` Méthode objet du tableau tabNoms
Au cas ou
    : (Evenement formulaire=Sur chargement)
        TOUT SELECTIONNER([Personnes])
        SELECTION VERS TABLEAU([Personnes]Noms;tabNoms;
                                                [Personnes]Numéro ID;tabIDs)
        TRIER TABLEAU(tabNoms;tabIDs;>)
    : (Evenement formulaire=Sur libération)
        EFFACER VARIABLE(tabNoms)
        EFFACER VARIABLE(tabIDs)
    : (Evenement formulaire=Sur clic)
        Si (tabNoms#0)
                ` Utiliser le tableau tabIDs pour récupérer le bon enregistrement
            CHERCHER([Personnes];[Personnes]Numéro IDr=tabIDs{tabNoms})
                ` Traiter ici l'enregistrement
        Fin de si
Fin de cas

    ` Méthode objet du bouton bAversZ
    ` Tri croissant des tableaux en conservant la synchronisation
TRIER TABLEAU(tabNoms;tabIDs;>)
```

```
      ` Méthode objet du bouton bZversA
      ` Tri décroissant des tableaux en conservant la synchronisation
   TRIER TABLEAU(tabNoms;tabIDs;<)
```

Référence

SELECTION VERS TABLEAU, TABLEAU MULTI TRI, TRIER.

VALEURS DISTINCTES (leChamp; tableau)

Paramètre	Type		Description
leChamp	Champ	→	Champ à utiliser
tableau	Tableau	←	Tableau devant recevoir les données du champ indexable

Description

VALEURS DISTINCTES crée et remplit le tableau tableau avec toutes les valeurs distinctes provenant du champ leChamp pour la sélection courante de la table à laquelle le champ appartient.

Vous pouvez passer à cette commande tout type de champ **indexable**, c'est-à-dire dont le type supporte l'indexation mais qui n'est pas forcément indexé. Toutefois, l'exécution de la commande avec des champs non indexés est plus lente qu'avec des champs indexés. A noter également que dans ce cas, la commande perd l'enregistrement courant.
VALEURS DISTINCTES analyse et extrait les valeurs distinctes pour les enregistrements sélectionnés uniquement.

Note : Lorsque vous exécutez VALEURS DISTINCTES au sein d'une transaction non encore terminée, la commande tient compte des enregistrements créés au cours de la transaction.

Le tableau utilisé par VALEURS DISTINCTES doit être du même type que le champ passé en premier paramètre, sinon le tableau est retypé. Il y a une exception à cette règle : si le champ est de type Heure, le tableau correspondant doit être de type Entier long.

Après l'appel, la taille du tableau est égale au nombre de valeurs distinctes trouvées dans la sélection. La commande ne modifie pas la sélection courante ni l'enregistrement courant. Les éléments dans tableau sont triés par ordre croissant car VALEURS DISTINCTES utilise l'index du champ. Si cet ordre vous convient, vous n'avez donc pas besoin d'appeler TRIER TABLEAU après l'exécution de VALEURS DISTINCTES.

Note : Lorsque VALEURS DISTINCTES est exécutée avec un champ texte associé à un index de mots-clés, la commande remplit le tableau avec les mots-clés de l'index. A la différence des autres types de données, les valeurs retournées diffèrent donc en fonction de l'existence de l'index. L'index de mots-clés est toujours pris en compte, même si le champ est également associé à un index standard.

ATTENTION : VALEURS DISTINCTES peut créer des tableaux de taille importante, en fonction de la taille de la sélection courante, ainsi que du type et de la taille des données à charger. Comme les tableaux résident en mémoire, il peut être utile de tester la taille des tableaux créés après l'exécution de la commande, ou d'utiliser une méthode projet d'interception d'erreurs installée par la commande APPELER SUR ERREUR.

4D Server : Cette commande est optimisée pour 4D Server. Le tableau est créé et les valeurs sont calculées sur le serveur. Seul le tableau est envoyé au client.

Exemple

L'exemple suivant crée une liste de villes à partir de la sélection courante et indique à l'utilisateur le nombre de villes dans lesquelles la société dispose de magasins :

```
TOUT SELECTIONNER([Revendeurs]) ` Créer une sélection d'enregistrements
VALEURS DISTINCTES([Revendeurs]Ville;taVilles)
ALERTE("Cette société dispose de magasins dans " +Chaine(Taille tableau(taVilles))+
                                                                    " villes.")
```

Référence

APPELER SUR ERREUR, SELECTION LIMITEE VERS TABLEAU, SELECTION VERS TABLEAU.

59

Transactions

Les transactions sont une série de modifications effectuées à l'intérieur d'un process sur des données reliées entre elles. Une transaction n'est sauvegardée de façon définitive dans la base que si la transaction est validée. Si une transaction n'est pas complétée, parce qu'elle est annulée ou en raison d'un quelconque événement extérieur, les modifications ne sont pas sauvegardées.

Pendant une transaction, toutes les modifications effectuées sur les données de la base dans le process sont stockées localement dans un buffer temporaire. Si la transaction est acceptée avec VALIDER TRANSACTION, les changements sont sauvegardés de façon définitive. Si la transaction est annulée avec ANNULER TRANSACTION, les changements ne sont pas sauvegardés. Dans tous les cas, ni la sélection courante ni l'enregistrement courant ne sont modifiés par les commandes de gestion des transactions.

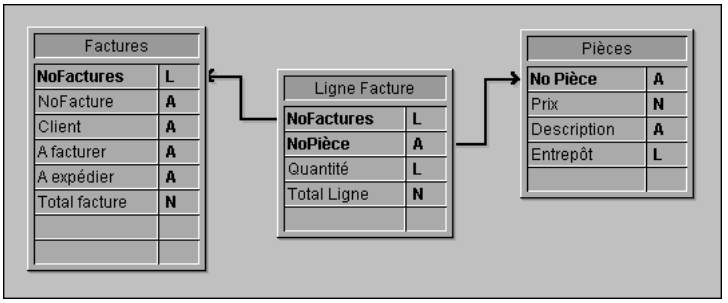
A compter de la version 11, 4D prend en charge les transactions imbriquées, c'est-à-dire les transactions sur plusieurs niveaux hiérarchiques. Le nombre de sous-transactions autorisées est illimité. La commande Niveau de la transaction permet de connaître le niveau courant de transaction dans lequel le code est exécuté.
Lorsque vous utilisez des transactions imbriquées, le résultat de chaque sous-transaction dépend de la validation ou de l'annulation de la transaction du niveau supérieur. Si la transaction supérieure est validée, les résultats des sous-transactions sont entérinés (validation ou annulation). En revanche, si la transaction supérieure est annulée, toutes les sous-transactions sont annulées, quels que soient leurs sous-résultats.

Option de compatibilité
Les transactions imbriquées pouvant générer des dysfonctionnements dans les bases développées avec des versions précédentes de 4D, il est désactivé par défaut dans les bases converties (les transactions restent limitées à un seul niveau). Si vous souhaitez bénéficier des transactions sur plusieurs niveaux dans une base convertie, vous devez l'indiquer explicitement en cochant l'option **Transactions imbriquées** dans la page "Application/Compatibilité" des Préférences de l'application.
Cette option n'apparaît que pour les bases de données converties. Par défaut, elle n'est pas cochée. Elle est spécique à chaque base de données. Elle n'a pas d'effet sur les transactions effectuées dans le moteur SQL de 4D. Les transactions SQL sont toujours multi-niveaux.

Exemples de transactions

L'exemple de cette section s'appuie sur la structure présentée ci-dessous. C'est une base relativement simple de facturation. Les lignes de factures sont stockées dans une table appelée [Ligne Facture], qui est reliée à la table [Factures] par une relation entre les champs [Factures]NoFacture et [Ligne Facture]NoFacture. Lorsqu'une facture est ajoutée, un numéro unique est calculé avec la commande Numerotation automatique. Le lien entre [Factures] et [Ligne Facture] est du type aller-retour automatique. L'option "Mise à jour auto dans les sous-formulaires" est cochée.

La lien entre [Ligne Facture] et [Pièces] est manuel.



Quand un utilisateur saisit une facture, les actions suivantes doivent être exécutées :
• Ajouter un enregistrement dans la table [Factures].
• Ajouter plusieurs enregistrements dans la table [Ligne Facture].
• Mettre à jour le champ [Pièces]Entrepôt pour chaque pièce figurant sur la facture.

En d'autres termes, vous devez sauvegarder les données liées. C'est la situation type où vous devez utiliser une transaction. Vous pourrez ainsi être certain de pouvoir soit sauvegarder tous ces enregistrements pendant l'opération, soit annuler la transaction si un enregistrement ne peut être ajouté ou mis à jour.

Si vous n'utilisez pas une transaction, vous ne pouvez pas garantir l'intégrité logique des données de votre base. Par exemple, si un enregistrement parmi ceux de la table [Pièces] est verrouillé, vous ne pourrez pas mettre à jour la quantité stockée dans le champ [Pièces]Entrepôt. Ce champ sera alors logiquement incorrect. La somme des pièces vendues et restantes dans l'entrepôt ne sera pas égale à la quantité d'origine saisie dans l'enregistrement. Vous pouvez éviter cette situation en utilisant les transactions.

Il y a plusieurs façons d'effectuer une saisie sous transaction :

(1) Vous pouvez gérer les transactions en utilisant les commandes de transaction DEBUT TRANSACTION, VALIDER TRANSACTION et ANNULER TRANSACTION. Vous pouvez par exemple écrire :

```
LECTURE ECRITURE([Ligne Facture])
LECTURE ECRITURE([Pièces])
FORMULAIRE ENTREE([Factures];"Saisie")
Repeter
    DEBUT TRANSACTION
    AJOUTER ENREGISTREMENT([Factures])
    Si (OK=1)
        VALIDER TRANSACTION
    Sinon
        ANNULER TRANSACTION
    Fin de si
Jusque (OK=0)
LECTURE SEULEMENT(*)
```

(2) Pour réduire les verrouillages des enregistrements pendant la saisie de données, vous pouvez aussi choisir de gérer les transactions à partir de la méthode du formulaire et d'accéder aux tables en LECTURE ECRITURE uniquement quand cela est nécessaire.
Vous effectuez la saisie de données en utilisant le formulaire de saisie pour [Factures], qui contient la table liée [Factures]Lignes dans un sous-formulaire. Le formulaire comporte deux boutons : bAnnuler et bOK. Aucune action ne leur est attribuée.

La boucle d'ajout devient alors :

```
LECTURE ECRITURE([Ligne Facture])
LECTURE SEULEMENT([Pièces])
FORMULAIRE ENTREE([Factures];"Input")
Repeter
    AJOUTER ENREGISTREMENT([Factures])
Jusque (bOK=0)
LECTURE SEULEMENT([Ligne Facture])
```

Notez que la table [Pièces] est désormais en "lecture seulement" pendant la saisie de données. L'accès en lecture/écriture ne s'active que si les données sont validées.

La transaction est ouverte dans la méthode du formulaire entrée de la table [Factures] :

```
Au cas ou
    : (Evenement formulaire=Sur chargement)
        DEBUT TRANSACTION
        [Factures]NoFactures:=Numerotation automatique([Factures]NoFactures)
    Sinon
        [Factures]Total facture:=Somme([Ligne Facture]Total ligne)
Fin de cas
```

Si vous cliquez sur le bouton bAnnuler, la saisie et la transaction doivent être annulées. Voici la méthode objet du bouton bAnnuler:

```
Au cas ou
    : (Evenement formulaire=Sur clic)
        ANNULER TRANSACTION
        NE PAS VALIDER
Fin de cas
```

Si vous cliquez sur le bouton bOK, la saisie et la transaction doivent être acceptées. Voici la méthode objet du bouton bOK :

```
Au cas ou
    : (Evenement formulaire=Sur clic)
        $NbLines:=Enregistrements trouves([Ligne Facture])
        LECTURE ECRITURE([Pièces])
            ` Passer en lecture/écriture pour accéder à la table [Pièces]
        DEBUT SELECTION([Ligne Facture])  ` Commencer à la première ligne
        $ValidTrans:=Vrai  ` Tout devrait marcher
        Boucle ($Line;1;$NbLines)  ` Pour chaque ligne
            CHARGER SUR LIEN([Ligne Facture]NoPiece)
            OK:=1  ` Vous voulez continuer
            Tant que (Enregistrement verrouille([Pièces]) & (OK=1))
                    ` Essayer d'obtenir l'enregistrement en lecture/écriture
                CONFIRMER("La pièce "+[Ligne Facture]NoPiece+" est utilisée. Vous
                                                                    attendez ?")
                Si (OK=1)
                    ENDORMIR PROCESS(Numero du process courant;60)
                    CHARGER ENREGISTREMENT([Pièces])
                Fin de si
            Fin tant que
            Si (OK=1)
                    ` Mettre à jour quantité dans l'entrepôt
                [Pièces]Entrepôt:=[Pièces]Entrepôt-[Ligne Facture]Quantité
                STOCKER ENREGISTREMENT([Pièces])  ` Sauvegarder l'enregistrement
```

```
            Sinon
                $Ligne:=$NbLines+1   ` Sortir de la boucle
                $ValidTrans:=Faux
            Fin de si
            ENREGISTREMENT SUIVANT([Ligne Facture])   ` Aller à la ligne suivante
        Fin de boucle
        LECTURE SEULEMENT([Pièces])   ` Mettre la table en mode lecture seulement
        Si ($ValidTrans)
            STOCKER ENREGISTREMENT([Factures])
                ` Sauvegarder les enregistrements
            VALIDER TRANSACTION   ` Valider toutes les modifications de la base
        Sinon
            ANNULER TRANSACTION   ` Tout annuler
        Fin de si
        NE PAS VALIDER   ` Quitter le formulaire
    Fin de cas
```

Dans le code ci-dessus, quel que soit le bouton sur lequel l'utilisateur a cliqué, nous appelons la commande NE PAS VALIDER. Le nouvel enregistrement n'est pas validé par un appel à VALIDER mais par STOCKER ENREGISTREMENT. De plus, vous remarquez que STOCKER ENREGISTREMENT est appelée juste avant la commande VALIDER TRANSACTION. Ainsi, la sauvegarde de l'enregistrement [Factures] est partie intégrante de la transaction. Appeler la commande VALIDER validerait aussi l'enregistrement mais dans ce cas, la transaction serait validée avant le stockage de la facture. Autrement dit, l'enregistrement serait sauvegardé en-dehors de la transaction.

En fonction de vos besoins, personnalisez votre base à votre convenance, comme dans les exemples précédents. Dans le dernier exemple, la gestion du verrouillage des enregistrements de la table [Pièces] pourrait être plus élaborée.

Référence

ANNULER TRANSACTION, DEBUT TRANSACTION, Transaction en cours, VALIDER TRANSACTION.

ANNULER TRANSACTION

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

ANNULER TRANSACTION annule la transaction ouverte par la commande DEBUT TRANSACTION de niveau correspondant dans le process courant. ANNULER TRANSACTION annule toutes les opérations éventuellement exécutées sur les données pendant la transaction.

Référence

DEBUT TRANSACTION, Niveau de la transaction, Transaction en cours, Utiliser des transactions, VALIDER TRANSACTION.

DEBUT TRANSACTION

version 11 (Modifiée)

DEBUT TRANSACTION

Paramètre	Type	Description

Cette commande ne requiert pas de paramètre

Description

DEBUT TRANSACTION débute une transaction dans le process courant. Toutes les modifications effectuées sur les données (enregistrements) de la base à l'intérieur de la transaction seront stockées temporairement jusqu'à ce que la transaction soit validée ou annulée.

A compter de la version 11 de 4D, vous pouvez imbriquer plusieurs transactions (sous-transactions). Chaque transaction ou sous-transaction doit être finalement annulée ou validée. A noter que si la transaction principale est annulée, toutes les sous-transactions sont annulées, quels que soient leurs résultats.

Référence

ANNULER TRANSACTION, Niveau de la transaction, Transaction en cours, Utiliser des transactions, VALIDER TRANSACTION.

Niveau de la transaction → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	Niveau de transaction courant (0 si aucune transaction n'a été démarrée)

Description

La commande Niveau de la transaction retourne le niveau de transaction courant pour le process. Cette commande prend en compte toutes les transactions du process courant, qu'elles aient été démarrées via le langage de 4D ou via le SQL.

Référence

DEBUT TRANSACTION, Transaction en cours, Utiliser des transactions.

Transaction en cours → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	VRAI si le process courant est en transaction, FAUX sinon

Description

La commande Transaction en cours retourne VRAI si le process courant est en transaction, sinon elle retourne FAUX.

Exemple

Si vous effectuez des opérations (ajout, modification ou suppression) sur de multiples enregistrements, vous pouvez rencontrer des enregistrements verrouillés. Dans ce cas, pour préserver l'intégrité des données, vous devez avoir ouvert une transaction, de manière à ce que vous puissiez faire “marche arrière” et annuler l'ensemble de l'opération depuis le début, sans que les données de la base soient modifiées.

Si vous effectuez l'opération depuis un trigger ou une sous-routine pouvant être appelé(e) dans une transaction ou hors transaction, l'utilisation de la commande Transaction en cours vous permet de vérifier que la méthode du process courant ou la méthode appelante a bien ouvert une transaction. Si ce n'est pas le cas, vous ne commencez même pas l'opération, car, en cas d'échec au cours du processus, vous ne pourriez pas revenir sur les opérations déjà effectuées.

Référence

ANNULER TRANSACTION, DEBUT TRANSACTION, Présentation des triggers, VALIDER TRANSACTION.

VALIDER TRANSACTION

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

VALIDER TRANSACTION accepte la transaction ouverte par la commande DEBUT TRANSACTION de niveau correspondant dans le process courant. VALIDER TRANSACTION sauvegarde toutes les modifications effectuées sur les données de la base pendant la transaction.

A compter de la version 11 de 4D, vous pouvez imbriquer plusieurs transactions (sous-transactions). Si la transaction principale est annulée, toutes les sous-transactions sont annulées, même si elles ont été validées individuellement à l'aide de cette commande.

Variables et ensembles système

La variable système OK prend la valeur 1 si la transaction a été correctement validée, sinon elle prend la valeur 0.

Référence

ANNULER TRANSACTION, DEBUT TRANSACTION, Transaction en cours, Utiliser des transactions.

60

Triggers

Un trigger est une méthode associée à une table. C'est une propriété d'une table. Vous n'appelez pas un trigger, les triggers sont appelés automatiquement par le moteur de 4D à chaque fois qu'un enregistrement de la table est manipulé (ajout, suppression et modification). Les triggers sont des méthodes qui peuvent éviter des opérations “illégales” dans votre base. Par exemple, dans une facturation, vous pouvez empêcher qu'un utilisateur crée une facture sans spécifier à qui elle doit être adressée. Les triggers sont un outil puissant permettant de contrôler les opérations sur les tables, et d'éviter des pertes de données accidentelles. Vous pouvez créer des triggers très simples et les rendre de plus en plus sophistiqués.

Activer et créer un trigger

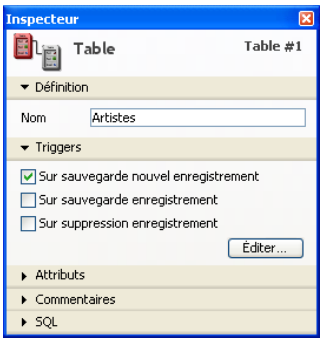
Par défaut, lorsque vous créez une table en mode Développement, la table n'a pas de trigger.

Pour utiliser un trigger pour une table, vous devez :
• activer le trigger et indiquer à 4D quand l'appeler.
• créer et écrire le code pour le trigger.

Activer un trigger qui n'est pas encore écrit ou écrire un trigger sans l'activer n'affecte pas les opérations effectuées sur une table.

1. Activer un Trigger

Pour activer le trigger, sélectionnez les options **Triggers** pour la table dans la fenêtre de l'Inspecteur de structure :



Voici la description de ces options :

Sur sauvegarde nouvel enregistrement
Si cette option est sélectionnée, le trigger sera appelé à chaque fois qu'un enregistrement est créé dans la table, c'est-à-dire dans les circonstances suivantes :
• Vous ajoutez un enregistrement lors de la saisie de données (mode Développement, commande AJOUTER ENREGISTREMENT ou commande SQL INSERT).
• Vous créez puis sauvegardez un enregistrement avec CREER ENREGISTREMENT et STOCKER ENREGISTREMENT. Notez que le trigger est appelé au moment où vous exécutez STOCKER ENREGISTREMENT, et non quand il est réellement créé.
• Vous importez des enregistrements (mode Développement ou commandes du langage).
• Vous appelez d'autres commandes qui créent et/ou sauvegardent de nouveaux enregistrements (par exemple TABLEAU VERS SELECTION, STOCKER SUR LIEN, etc.).
• Vous utilisez un plug-in 4D qui appelle les commandes CREER ENREGISTREMENT et STOCKER ENREGISTREMENT.

Sur sauvegarde enregistrement
Si cette option est sélectionnée, le trigger sera appelé à chaque fois qu'un enregistrement de la table est modifié, c'est-à-dire dans les circonstances suivantes :
• Vous modifiez un enregistrement en saisie de données (mode Développement, commande MODIFIER ENREGISTREMENT ou commande SQL UPDATE)
• Vous sauvegardez un enregistrement existant avec STOCKER ENREGISTREMENT.
• Vous appelez une commande qui provoque la sauvegarde d'un enregistrement existant (par exemple, TABLEAU VERS SELECTION, APPLIQUER A SELECTION, etc.)
• Vous utilisez un plug-in 4D qui appelle la commande STOCKER ENREGISTREMENT.

Sur suppression enregistrement
Si cette option est sélectionnée, le trigger sera appelé à chaque fois qu'un enregistrement de la table est supprimé, c'est-à-dire dans les circonstances suivantes :
• Vous supprimez un enregistrement en mode Développement ou en appelant la commande SUPPRIMER ENREGISTREMENT, SUPPRIMER SELECTION ou la commande SQL DELETE.
• Vous effectuez des opérations qui provoquent la suppression d'un enregistrement lié par l'intermédiaire des options de contrôle de suppression d'un lien.
• Vous utilisez un plug-in 4D qui appelle la commande SUPPRIMER ENREGISTREMENT.

Note : La commande VIDER TABLE ne provoque PAS l'appel du trigger.

2. Créer un trigger
Pour créer un trigger pour une table, utilisez la fenêtre de l'Explorateur, cliquez sur le bouton **Editer** dans la palette de l'Inspecteur de structure ou double-cliquez sur le titre de la table dans la fenêtre de Structure en appuyant sur la touche **Alt** (sous Windows) ou **Option** (sous Mac OS). Pour plus d'informations, reportez-vous au manuel *Mode Développement*.

Evénements moteur

Un trigger peut être invoqué pour l'un des trois événements moteur décrits ci-dessus. Dans le trigger, vous détectez quel événement a lieu en appelant la fonction Evenement moteur. Cette fonction retourne une valeur numérique qui indique l'événement moteur.

Typiquement, vous écrivez un trigger avec une structure du type "Au cas ou" sur le résultat retourné par Evenement moteur :

```
    ` Trigger pour [UneTable]
C_ENTIER LONG($0)
$0:=0   ` On suppose que la requête est acceptée
Au cas ou
    : (Evenement moteur=Sur sauvegarde nouvel enreg)
        ` Effectuer les actions appropriées pour sauvegarder l'enregistrement créé.
    : (Evenement moteur=Sur sauvegarde enregistrement)
        ` Effectuer les actions appropriées pour sauvegarder l'enregistrement déjà existant.
    : (Evenement moteur=Sur suppression enregistrement)
        ` Effectuer les actions appropriées pour détruire l'enregistrement.
Fin de cas
```

Les triggers sont des fonctions

Un trigger a deux finalités :
• Effectuer des actions sur l'enregistrement juste avant qu'il soit sauvegardé ou supprimé.
• Accepter ou rejeter une opération de base de données.

1. Effectuer des actions
A chaque fois qu'un enregistrement est sauvegardé (ajouté ou modifié) dans une table [Documents], vous souhaitez “estampiller ” l'enregistrement avec des marqueurs de création et de modification. Vous pouvez écrire le trigger suivant :

```
    ` Trigger pour table [Documents]
Au cas ou
    : (Evenement moteur=Sur sauvegarde nouvel enreg)
        [Documents]Creation Stamp:=Time stamp
        [Documents]Modification Stamp:=Time stamp
    : (Evenement moteur=Sur sauvegarde enregistrement)
        [Documents]Modification Stamp:=Time stamp
Fin de cas
```

Note : La fonction Time stamp utilisée dans cet exemple est une petite méthode projet retournant le nombre de secondes écoulées depuis une date choisie arbitrairement.

Une fois que ce trigger a été écrit et activé, peu importe la façon dont vous ajoutez ou modifiez un enregistrement dans la table [Documents] (saisie de données, import, méthode projet, plug-in 4D), la valeur des deux champs [Documents]Creation Stamp et [Documents]Modification Stamp sera automatiquement affectée par le trigger avant que l'enregistrement ne soit écrit sur disque.

Note : Voir l'exemple de la commande PROPRIETES DOCUMENT pour une analyse complète de cet exemple.

2. Accepter ou rejeter l'opération de la base
Pour accepter ou rejeter une opération de la base, le trigger doit retourner un **code d'erreur de trigger** dans le résultat de la fonction $0.

Exemple
Prenons le cas d'une table [Employés]. Pendant la saisie de données, vous contrôlez le champ [Employés]No Séc.Soc. Par exemple, lorsque l'utilisateur clique sur le bouton de validation, vous vérifiez le champ utilisant la méthode objet du bouton :

```
    ` Méthode objet bouton bAccept
Si (Bon No Séc.Soc ([Employés]No Séc.Soc))
    VALIDER
Sinon
    BEEP
    ALERTE ("Saisissez un numéro de sécurité sociale et cliquez de nouveau sur OK.")
Fin de si
```

Si la valeur du champ est correcte, vous acceptez la saisie de données, sinon vous affichez une alerte et restez en saisie de données.

Si vous créez aussi des enregistrements pour la table [Employés] par programmation, le code ci-dessous serait valide MAIS violerait la règle imposée dans la méthode objet créée plus haut :

```
    ` Extrait d'une méthode projet
    ` ...
CREER ENREGISTREMENT ([Employés])
[Employés]Nom :="DOE"
STOCKER ENREGISTREMENT ([Employés]) ` <- violation de la règle ! Il n'y a pas de
                                                    numéro de sécurité sociale !
```

En utilisant un trigger pour la table [Employés], vous pouvez appliquer la contrainte sur [Employés]No Séc.Soc à tous les niveaux de la base. Le trigger serait du type :

```
    ` Trigger pour [Employés]
$0:=0
$dbEvent:=Evenement moteur
Au cas ou
    : (($dbEvent=Sur sauvegarde nouvel enreg) | ($dbEvent=
                                                Sur sauvegarde enregistrement))
    Si (Non(Bon No Séc.Soc ([Employés]No Séc.Soc)))
        $0:=-15050
    Sinon
        ` ...
    Fin de si
        ` ...
Fin de cas
```

Une fois que ce trigger est écrit et activé, la ligne STOCKER ENREGISTREMENT([Employés]) de la méthode projet ci-dessus génèrera une erreur moteur -15050 et l'enregistrement ne sera PAS sauvegardé.

De la même façon, si un plug-in 4D essayait de sauvegarder un enregistrement dans [Employés] avec un numéro de sécurité sociale incorrect, le trigger génèrerait la même erreur et l'enregistrement ne serait pas sauvegardé non plus.

Le trigger garantit que personne (utilisateur, développeur, plug-in, 4D Open client avec 4D Server) ne peut violer la règle sur le numéro de sécurité sociale (à dessein ou par erreur).

Notez que même si vous n'avez pas créé de trigger pour une table, la base peut retourner des erreurs moteur lorsque vous essayez de sauvegarder ou de détruire un enregistrement. Vous pouvez, par exemple, recevoir l'erreur -9998, si vous essayez de sauvegarder un enregistrement.

Les triggers retournent de nouveaux types d'erreurs dans 4D :
• 4D gère les erreurs “normales” : index unique, contrôles relationnels, etc.
• En utilisant les triggers, vous pouvez créer des codes d'erreurs propres au contenu de votre application.

Important : Vous pouvez retourner le code d'erreur de votre choix. Cependant, n'utilisez pas des codes d'erreurs déjà utilisés par le moteur de 4D. Nous vous recommandons fortement d'utiliser des codes compris entre -32000 et -15000. Nous réservons les erreurs supérieures à -15000 au moteur de 4D.

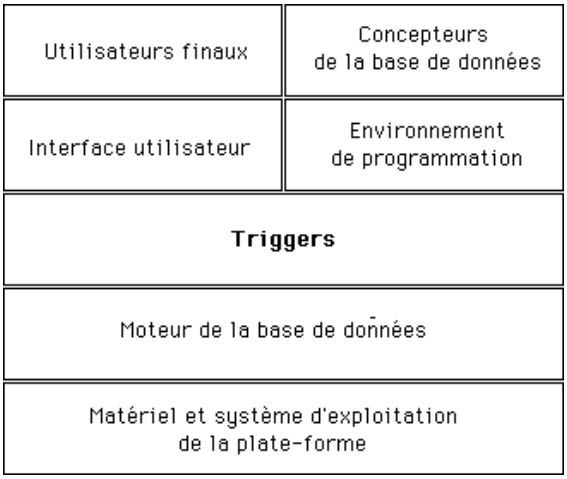
Au niveau du process, vous gérez les erreurs trigger de la même façon que les erreurs du moteur de base de données :
• vous pouvez laisser 4D afficher la boîte de dialogue standard d'erreur, la méthode est alors interrompue.
• vous pouvez utiliser une méthode de gestion d'erreur installée par APPELER SUR ERREUR et traiter l'erreur de façon appropriée.

Notes :
• Pendant la saisie, si une erreur trigger est retournée au moment où vous essayez de valider ou de supprimer un enregistrement, l'erreur est gérée comme une erreur sur un index unique. La boîte de dialogue d'erreur est affichée et vous restez en saisie de données. Même si vous n'utilisez une base qu'en mode Développement (et non en Application), vous bénéficiez des triggers.
• Lorsqu'une erreur est générée par un trigger dans le cadre d'une commande agissant sur une sélection d'enregistrements (telle que SUPPRIMER SELECTION), l'exécution de la commande est immédiatement stoppée, sans que la sélection ait été nécessairement traitée en totalité. Ce cas requiert de la part du développeur une gestion appropriée, basée par exemple sur la conservation temporaire de la sélection, le traitement et l'élimination de l'erreur avant l'exécution du trigger, etc.

Même si un trigger ne retourne pas d'erreur ($0:=0), cela ne signifie pas qu'une opération de la base s'effectuera correctement. Il peut y avoir eu un doublon sur l'index unique. Si l'opération est la mise à jour d'un enregistrement, ce dernier peut être verrouillé, une erreur d'entrée/sortie peut se produire, bien d'autres choses encore peuvent arriver. Ces vérifications sont effectuées après l'exécution du trigger. Cependant, du point de vue du plus haut niveau du process en exécution, les erreurs retournées par le moteur de la base de données ou celle d'un trigger sont de même nature : une erreur trigger est une erreur du moteur de la base de données.

Les triggers et l'architecture 4D

Les triggers sont exécutés au niveau du moteur de la base de données. Ce point est illustré dans le schéma suivant :



Les triggers sont exécutés **sur la machine où est situé le moteur de la base de données**. Si ce point est une évidence dans le cas de 4D en local, il convient de rappeler que pour 4D Server, les triggers sont exécutés sur la machine serveur (dans le process "jumeau" du process ayant déclenché le trigger) et non sur la machine cliente.

Quand un trigger est appelé, il s'exécute dans le contexte du process qui tente l'opération. Ci-dessous, ce process est appelé **process appelant** l'exécution du trigger.
Les éléments inclus dans ce contexte diffèrent suivant que la base est exécutée avec 4D en mode local ou avec 4D Server :
• avec 4D en mode local, le trigger fonctionne avec les sélections courantes, les enregistrements courants, les statuts lecture/écriture des tables, les verrouillages d'enregistrements, etc., du process appelant.
• avec 4D Server, seul le contexte de base de données du process client appelant est préservé (verrouillages d'enregistrements, statut lecture/écriture des tables...). 4D Server garantit également que l'enregistrement courant de la table du trigger est correctement positionné. Les autres éléments contextuels (sélections courantes par exemple) sont ceux du process du trigger.

Soyez prudent lorsque vous utilisez les autres objets de la base et du langage, car un trigger peut s'exécuter sur une machine différente de celle du process appelant : c'est le cas avec 4D Server !

• **Variables interprocess** : Un trigger a accès aux variables interprocess de la machine où il est exécuté. Avec 4D Server, ce peut être une machine différente de celle du process appelant.
• **Variables process** : Chaque trigger possède sa propre table de variables process. Un trigger n'a pas accès aux variables process du process appelant.
• **Variables locales** : Vous pouvez utiliser des variables locales dans un trigger. Leur aire d'action est l'exécution du trigger (elles sont créées/détruites au cours de cette exécution).
• **Sémaphores** : Un trigger peut tester ou placer des sémaphores globaux et locaux (sur la machine où il s'exécute dans ce dernier cas). Cependant, un trigger doit s'exécuter rapidement. En conséquence, utilisez plutôt des sémaphores locaux dans un trigger, sauf si vous avez une idée précise en tête.
• **Ensembles et sélections temporaires** : Si vous utilisez un ensemble ou une sélection temporaire dans un trigger, vous travaillez alors avec ceux de la machine où les triggers s'exécutent. En client/serveur, les ensembles et sélections temporaires "process" (dont le nom ne débute ni par $ ni par <>) créés sur le client sont visibles dans un trigger.
• **Interface utilisateur** : N'utilisez PAS d'éléments d'interface utilisateur dans un trigger (alerte, message ou dialogue). Cela signifie également que tracer le trigger dans la fenêtre du Débogueur doit être limité. Souvenez-vous que les triggers en client/serveur s'exécutent sur la machine 4D Server. Un message d'alerte affiché sur le poste serveur ne dit pas grand chose à l'utilisateur qui, lui, travaille sur sa machine cliente. Laissez le process appelant gérer l'interface utilisateur.

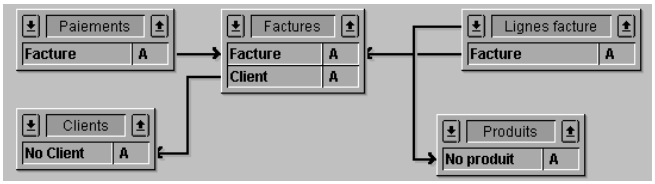
Triggers et transactions

Les transactions doivent être gérées au niveau du process appelant. Il est fortement déconseillé de gérer des transactions au niveau du trigger. Si, pendant l'exécution d'un trigger, vous devez ajouter, modifier ou détruire plusieurs enregistrements et souhaitez garantir l'intégrité de vos données à l'aide d'une transaction, vous devez d'abord tester (à partir du trigger) si le process appelant est en cours de transaction avec la commande Transaction en cours. En effet, si ce n'est pas le cas et si le trigger rencontre un enregistrement verrouillé, le process appelant n'aura aucun moyen d'annuler a posteriori les actions déjà effectuées par le trigger. Par conséquent, si vous n'êtes pas en transaction, ne commencez pas les opérations à exécuter, et retournez simplement une erreur dans $0 afin de signaler au process appelant que l'opération de base de données doit être exécutée dans une transaction.

Note : Afin d'optimiser le fonctionnement combiné des triggers et des transactions, 4D n'appelle PAS les triggers lors d'un VALIDER TRANSACTION. Cela évite que les triggers soient exécutés deux fois.

Triggers en cascade

Prenons l'exemple de la structure suivante :



Note : Les tables ont été contractées (il y a davantage de champs).

Pour les nécessités de cette documentation, nous admettrons que la base “autorise” la suppression d'une facture. Voyons aussi comment une telle opération serait gérée au niveau du trigger (puisque vous pourriez aussi décider d'effectuer l'opération au niveau du process).

Afin que soit maintenue l'intégrité relationnelle des données, la suppression d'une facture requiert les actions suivantes de la part du trigger de [Factures] :
• Décrémenter le champ Ventes de la table [Clients] du montant de la facture.
• Supprimer tous les enregistrements de [Lignes facture] liés à la facture.
• Ceci implique aussi que le trigger de [Lignes facture] décrémente le champ Quantité vendue des enregistrements [Produits] liés à la ligne de facture que l'on s'apprête à supprimer.
• Supprimer tous les enregistrements de [Paiements] liés à la facture.

Tout d'abord, le trigger de [Factures] ne doit effectuer ces actions que si le process appelant est en transaction, afin qu'une annulation rétroactive soit possible en cas de rencontre d'un enregistrement verrouillé.

Deuxièmement, le trigger de [Lignes facture] est **en cascade** avec le trigger de [Factures]. Le premier s'exécute “à l'intérieur” du second parce que la destruction des éléments de la liste est consécutive à un appel à SUPPRIMER SELECTION dans le trigger de [Factures].

Ajoutons que toutes les tables dans cet exemple ont des triggers activés pour tous les événements de la base de données. La cascade des triggers sera :

- Le trigger de Factures est appelé car le process appelant supprime une facture
  - Le trigger de Clients est appelé car le trigger Factures met à jour le champ Ventes
  - Le trigger de Lignes facture est appelé car le trigger Factures supprime une ligne (ce qui est répété)
    - Le trigger de Produits est appelé car le trigger Lignes facture met à jour le champ Quantité vendue
  - Le trigger de Paiements est appelé car le trigger Factures supprime un paiement (ce qui est répété)

Dans cette cascade, le trigger de [Factures] s'exécute au niveau 1, les triggers [Clients], [Lignes facture] et [Paiements] au niveau 2 et le trigger [Produits] au niveau 3.

Dans les triggers, vous pouvez détecter à quel niveau un trigger est exécuté grâce à la commande Niveau du trigger. De plus, vous pouvez aussi obtenir des informations sur les autres niveaux en utilisant la commande PROPRIETES DU TRIGGER.

Si, par exemple, vous détruisiez un enregistrement [Produits] à un niveau process, le trigger de [Produits] s'exécuterait au niveau 1, non au niveau 3, comme plus haut.

Avec Niveau du trigger et PROPRIETES DU TRIGGER, vous pouvez identifier la raison d'une action. Dans l'exemple ci-dessus, une facture est supprimée au niveau process. Prenons pour hypothèse que nous voulons détruire un enregistrement [Clients] au niveau process. Le trigger de [Clients] devrait alors être conçu pour détruire toutes les factures liées à ce client. Cela signifie que le trigger [Factures] devrait être invoqué comme plus haut, mais pour une autre raison. Du trigger [Factures], vous pouvez détecter si le niveau est 1 ou 2. S'il est 2, vous pouvez vérifier si oui ou non c'est à cause de la suppression de l'enregistrement Client lui-même. Si tel est le cas, vous n'avez même plus besoin de vous préoccuper de la mise à jour du champ Ventes.

Utilisation de la numérotation automatique dans un trigger

Quand vous manipulez l'événement moteur Sur sauvegarde nouvel enreg, vous pouvez appeler la commande Numerotation automatique pour maintenir un numéro d'identification unique pour chaque enregistrement d'une table. Par exemple :

```
    ` Trigger pour la table [Factures]
Au cas ou
    : (Evenement moteur=Sur sauvegarde nouvel enreg)
            ` ...
        [Factures]Facture Identification:=Numerotation automatique ([Factures])
            ` ...
Fin de cas
```

Référence

Evenement moteur, Méthodes, Niveau du trigger, Numero enregistrement, PROPRIETES DU TRIGGER.

Evenement moteur

Evenement moteur → Entier long

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Entier long	←	0 Hors de tout événement de trigger
			1 Sauvegarde d'un nouvel enregistrement
			2 Sauvegarde d'un enregistrement existant
			3 Suppression d'un enregistrement

Description

La commande Evenement moteur est appelée dans un trigger et renvoie une valeur numérique qui indique le type de l'événement de la base, ou la raison pour laquelle le trigger a été appelé.

4D fournit les constantes prédéfinies suivantes :

Constante	Type	Valeur
Sur sauvegarde nouvel enreg	Entier long	1
Sur sauvegarde enregistrement	Entier long	2
Sur suppression enregistrement	Entier long	3

Si, dans un trigger, vous effectuez des opérations de base de données sur plusieurs enregistrements (par exemple mise à jour de plusieurs enregistrements dans la table [Produits] et ajout d'enregistrement dans la table [Factures]), vous pouvez rencontrer des situations (comme des enregistrements verrouillés) qui empêchent le trigger d'exécuter correctement les opérations pour lesquelles il est appelé. Il vous faut alors stopper les actions de la base et retourner une erreur pour que le process appelant sache que la requête n'a pu être exécutée. Ce process doit également être en mesure d'annuler les opérations non exécutées. Autrement dit, lorsqu'une telle situation se produit, vous avez besoin de savoir dans le trigger si vous êtes en transaction avant même d'essayer de faire quoi que ce soit. Pour cela, utilisez la fonction Transaction en cours.

Dans 4D, il n'y a pas de limite, à part la mémoire disponible, aux appels de triggers en cascade. Pour optimiser l'exécution d'un trigger, vous pouvez écrire le code de vos triggers non seulement en fonction de l'événement de la base mais aussi du niveau de l'appel lorsque les triggers sont appelés en cascade. Par exemple, pendant l'événement de la base Suppression enregistrement pour la table [Factures], vous pouvez ne pas effectuer la mise à jour du champ [Clients]Ventes si la suppression de l'enregistrement de la table [Factures] fait partie de la suppression en cascade des factures liées à l'enregistrement dans la table [Clients] que vous êtes en train de supprimer. Pour cela, utilisez les routines Niveau du trigger et PROPRIETES DU TRIGGER.

Exemple

Utilisez la fonction Evenement moteur pour structurer vos triggers comme ci-dessous :

```
      ` Un trigger de la table [toute table]

   C_ENTIER LONG($0)
   $0:=0  ` S'assurer que la requête de la base sera accordée
   Au cas ou
      : (Evenement moteur=Sur sauvegarde nouvel enreg)
         ` Exécuter les actions appropriées pour la sauvegarde d'un nouvel enregistrement
      : (Evenement moteur=Sur sauvegarde enregistrement)
         ` Exécuter les actions appropriées pour la sauvegarde d'un enregistrement existant
      : (Evenement moteur=Sur suppression enregistrement)
         ` Exécuter les actions appropriées pour la suppression d'un enregistrement
   Fin de cas
```

Référence

Niveau du trigger, Présentation des triggers, PROPRIETES DU TRIGGER, Transaction en cours.

Niveau du trigger

Niveau du trigger → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		
Résultat	Numérique ←	Niveau d'exécution du trigger (0 si hors du cycle d'exécution du trigger)

Description

La commande Niveau du trigger retourne le niveau d'exécution du trigger.

Reportez-vous à la description des triggers en cascade dans la section Présentation des triggers.

Référence

Evenement moteur, Présentation des triggers, PROPRIETES DU TRIGGER.

PROPRIETES DU TRIGGER

version 11.2 (Modifiée)

PROPRIETES DU TRIGGER (niveauTrigger; evenementBase; tableNum; enregNum)

Paramètre	Type		Description
niveauTrigger	Numérique	→	Niveau d'exécution du trigger
evenementBase	Numérique	←	Evénement de base de données
tableNum	Numérique	←	Numéro de la table
enregNum	Numérique	←	Numéro de l'enregistrement

Description

La commande PROPRIETES DU TRIGGER fournit des informations sur le niveau d'exécution du trigger que vous avez passé dans niveauTrigger. Vous devez utiliser conjointement PROPRIETES DU TRIGGER et Niveau du trigger pour effectuer différentes actions en fonction de la cascade du trigger. Reportez-vous à la description des triggers en cascade dans la section Présentation des triggers.

Si vous passez un niveau d'exécution de trigger inexistant, la commande retourne 0 (zéro) dans chaque paramètre.

La nature de l'événement de base de données pour le niveau d'exécution du trigger est retournée dans evenementBase. Les constantes prédéfinies suivantes sont fournies :

Constante	Type	Valeur
Sur sauvegarde nouvel enreg	Entier long	1
Sur sauvegarde enregistrement	Entier long	2
Sur suppression enregistrement	Entier long	3

Le numéro de table et d'enregistrement pour l'enregistrement concerné par l'événement de base de données pour le niveau d'exécution du trigger sont retournés dans tableNum et enregNum.

Référence

A propos des numéros d'enregistrements, Evenement moteur, Niveau du trigger, Présentation des triggers.

61

Utilisateurs et groupes

Appartient au groupe (nomUtilisateur; groupe) → Booléen

Paramètre	Type		Description
nomUtilisateur	Alpha	→	Nom de l'utilisateur
groupe	Alpha	→	Nom du groupe
Résultat	Booléen	←	Vrai = utilisateur est dans groupe Faux = utilisateur n'est pas dans groupe

Description

La fonction Appartient au groupe retourne Vrai si utilisateur appartient au groupe.

Exemple

L'exemple suivant recherche des factures. Si l'utilisateur courant est dans le groupe Administration, il pourra accéder aux formulaires qui affichent des informations confidentielles. Sinon, des formulaires standard sont affichés :

```
CHERCHER([Factures];[Factures]Prix>100)
Si (Appartient au groupe(Utilisateur courant;"Administration")
    FORMULAIRE SORTIE([Factures];"Confidentiel_Sortie")
    FORMULAIRE ENTREE([Factures];"Conf_Saisie")
Sinon
    FORMULAIRE SORTIE([Factures];"Sortie_Standard")
    FORMULAIRE ENTREE([Factures];"Entrée_Standard")
Fin de si
MODIFIER SELECTION([Factures];*)
```

Référence

Utilisateur courant.

BLOB VERS UTILISATEURS (utilisateurs)

Paramètre	Type		Description
utilisateurs	BLOB	→	BLOB (crypté) contenant des comptes utilisateurs créés et sauvegardés par l'Administrateur

Description

La commande BLOB VERS UTILISATEURS ajoute dans la base de données les comptes utilisateurs présents dans le BLOB utilisateurs. Le BLOB utilisateurs est crypté et doit impérativement avoir été créé par la commande UTILISATEURS VERS BLOB.

Seuls l'Administrateur et le Super_Utilisateur de la base peuvent exécuter cette commande. Si un autre utilisateur tente de l'exécuter, la commande ne fait rien et une erreur de privilège (-9949) est générée.

L'ajout de chaque compte utilisateur est effectué en tenant compte des règles suivantes :
• Le numéro d'identification de l'utilisateur sert de référence. Les utilisateurs sont traités dans l'ordre des numéros.
• Si le numéro existe déjà dans la structure de la base, les informations du compte sont mises à jour si nécessaire en fonction des informations contenues dans le BLOB.
• Si le numéro n'existe pas dans la structure de la base, l'utilisateur est créé en fonction des informations contenues dans le BLOB.
• Si le numéro correspond à un compte utilisateur supprimé dans la structure de la base, le compte est mis à jour en fonction des informations contenues dans le BLOB.
• Si les informations contenues dans le BLOB indiquent que le compte utilisateur est supprimé, le compte est supprimé dans la structure de la base.
• Les utilisateurs mis à jour sont associés aux groupes suivant les informations du BLOB.
• Si un groupe n'existe pas, il est ajouté.

Si la commande est exécutée correctement, la variable système *OK* prend la valeur 1, sinon elle prend la valeur 0.

Note de compatibilité : Les fichiers d'utilisateurs et groupes (extension .4UG) créés par la commande de menu **Enregistrer les groupes & utilisateurs...** dans une version de 4D antérieure à la 2004 peuvent être chargés dans 4D version 2004 et suivantes via ces instructions :

```
DOCUMENT VERS BLOB(mondoc; blob)
BLOB VERS UTILISATEURS(blob)
```

En revanche, les fichiers d'utilisateurs et groupes générés dans 4D à partir de la version 2004 ne peuvent pas être ouverts avec une version précédente.

Référence

UTILISATEURS VERS BLOB.

Variables et ensembles système

Si la commande est exécutée correctement, la variable système *OK* prend la valeur 1, sinon elle prend la valeur 0.

CHANGER LICENCES

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

La commande CHANGER LICENCES affiche la boîte de dialogue de mise à jour des licences 4D.
Cette boîte de dialogue vous permet d'activer des plug-ins, le serveur Web ou, avec 4D Server, d'ajouter des numéros d'expansion afin d'accroître le nombre de clients pouvant utiliser simultanément la base et ses plug-ins.

Note : Dans 4D et 4D Server, vous affichez cette boîte de dialogue en sélectionnant la commande **Mise à jour des licences...** dans le menu **Aide**.

CHANGER LICENCES permet d'activer des licences et d'ajouter des numéros d'expansion dans une application compilée diffusée à vos clients. Les développeurs 4D et les administrateurs de systèmes peuvent utiliser cette commande pour diffuser une application 4D, en laissant à leurs clients le soin de saisir eux-même les numéros sans devoir leur faire parvenir une mise à jour de l'application.

Pour plus d'informations sur le fonctionnement de cette boîte de dialogue, reportez-vous au Guide d'installation de 4D.

Exemple

Dans une boîte de dialogue de configuration ou de préférences personnalisée, vous placez un bouton auquel la méthode suivante est associée :

```
` Méthode objet du bouton bLicence
CHANGER LICENCES
```

Vous permettrez ainsi à l'utilisateur d'activer des licences sans avoir à modifier la base de données.

CHANGER MOT DE PASSE (motDePasse)

Paramètre	Type		Description
motDePasse	Alpha	→	Nouveau mot de passe

Description

CHANGER MOT DE PASSE permet de changer le mot de passe de l'utilisateur courant. Cette commande remplace le mot de passe courant par le nouveau mot de passe que vous passez dans motDePasse.

Attention : Les mots de passe différencient les caractères majuscules et minuscules.

Exemple

L'exemple suivant permet à l'utilisateur de modifier son mot de passe :

```
CHANGER UTILISATEUR COURANT  ` Afficher la boîte de dialogue des mots de passe
Si (OK=1)
    $pw1:=Demander("Saisissez le nouveau mot de passe pour "+Utilisateur courant)
        ` Le mot de passe doit comporter au moins cinq caractères
    Si (((OK=1) & ($pw1#"")) & (Longueur($pw1)>5))
            ` Vérifier qu'un mot de passe valide a été saisi
        $pw2:=Demander("Saisissez de nouveau le mot de passe")
        Si ((OK=1) & ($pw1=$pw2))
            CHANGER MOT DE PASSE($pw2)  ` Modifier le mot de passe
        Fin de si
    Fin de si
Fin de si
```

Référence

CHANGER JEU DE CARACTERES, CHANGER UTILISATEUR COURANT.

CHANGER PRIVILEGES

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Description

CHANGER PRIVILEGES permet de modifier le système de mots de passe. Lorsque cette commande est exécutée, la fenêtre de la boîte à outils de 4D contenant les pages Utilisateurs et Groupes est appelée pour modifier les privilèges.

Note : Cette commande ouvre une fenêtre modale. Par conséquent, vous ne devez pas l'appeler depuis une autre fenêtre modale, sinon l'ouverture de la fenêtre sera impossible et la commande ne fera rien.

Les groupes peuvent être modifiés par le Super_Utilisateur et l'Administrateur et par les propriétaires de groupe. Seuls le Super_Utilisateur et l'Administrateur peuvent modifier tous les groupes. Les propriétaires de groupe ne peuvent modifier que leur propre groupe. Des utilisateurs peuvent être ajoutés et retirés des groupes. Cette commande ne fait rien si aucun groupe n'est défini.

Le Super_Utilisateur et l'Administrateur peuvent créer des utilisateurs et les placer dans des groupes.

Exemple

L'exemple suivant affiche la fenêtre de gestion des utilisateur et des groupes :

```
CHANGER PRIVILEGES
```

Référence

CHANGER MOT DE PASSE, CHANGER UTILISATEUR COURANT.

CHANGER UTILISATEUR COURANT

CHANGER UTILISATEUR COURANT {(utilisateur; motDePasse)}

Paramètre	Type		Description
utilisateur	Alpha | Num	→	Nom ou Numéro de référence unique de l'utilisateur
motDePasse	Alpha	→	Mot de passe (non crypté)

Description

CHANGER UTILISATEUR COURANT permet de changer l'identité de l'utilisateur courant dans la base, sans devoir la quitter. Le changement d'identité peut être effectué par l'utilisateur lui-même via la boîte de dialogue de connexion à la base (lorsque la commande est appelée sans paramètres) ou directement par la commande. Lorsqu'il change d'identité, l'utilisateur abandonne ses anciens privilèges au profit de ceux de l'utilisateur choisi.

Le nouveau compte est utilisé pour tous les process “utilisateurs” courants de la base. A noter qu'un process créé par un autre process hérite du compte utilisateur de celui-ci.

4D Server : Sur le poste serveur, les process démarrés via les méthodes bases Sur démarrage serveur, Sur arrêt serveur, etc., sont exécutés sous le compte du Super_Utilisateur.

Si la commande CHANGER UTILISATEUR COURANT est exécutée sans paramètres, la boîte de dialogue de connexion à la base s'affiche. L'utilisateur doit alors saisir ou sélectionner un nom et un mot de passe valides pour entrer dans la base. Le contenu de la boîte de dialogue de connexion dépend des options définies dans la page **Application/Accès** des Préférences de la base.

Vous pouvez également passer les deux paramètres facultatifs utilisateur et motDePasse afin de spécifier par programmation le nouveau compte à utiliser.
Passez dans le paramètre utilisateur le nom ou le numéro de référence unique (réfUtilisateur) du compte à utiliser. Les noms et les numéros des utilisateurs peuvent être obtenus via la commande LIRE LISTE UTILISATEURS.

Numéro de référence de l'utilisateur	Description de l'utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le deuxième, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le deuxième, et ainsi de suite).

Si le compte d'utilisateur désigné n'existe pas ou a été supprimé, l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR. Sinon, vous pouvez appeler la fonction Utilisateur supprime pour tester le compte utilisateur avant d'appeler cette commande.

Passez dans le paramètre motDePasse le mot de passe non crypté du compte de l'utilisateur. Si le mot de passe ne correspond pas à l'utilisateur, la commande ne fait rien et l'erreur -9978 est générée.

La commande est temporisée afin d'éviter des attaques par "force brute" (essais automatiques de multiples combinaisons de noms d'utilisateurs/mots de passe). Ainsi, au bout du quatrième appel à la commande, elle n'est exécutée qu'après 10 secondes d'attente. Cette temporisation est globale au poste de travail.

Proposer une boîte de dialogue de gestion d'accès personnalisée

La commande CHANGER UTILISATEUR COURANT permet de mettre en place des boîtes de dialogue personnalisées pour la saisie du nom et du mot de passe (avec règles de saisie et d'expiration) tout en bénéficiant des avantages du système intégré de contrôle des accès de 4D. Le principe est le suivant :

1. L'entrée dans la base s'effectue directement en mode "Utilisateur par défaut", sans boîte de dialogue.

2. Dans la Méthode base Sur ouverture, le développeur provoque l'affichage d'une boîte de dialogue personnalisée de saisie du nom d'utilisateur et du mot de passe (à l'aide de la commande DIALOGUE ou AJOUTER ENREGISTREMENT par exemple). Tout type de traitement peut être envisagé dans la boîte de dialogue :

- Il est possible d'afficher la liste des utilisateurs de la base, comme dans la boîte de dialogue d'accès standard de 4D, à l'aide de la commande LIRE LISTE UTILISATEURS.
- Le champ de saisie du mot de passe peut contenir divers contrôles afin de vérifier la validité des caractères saisis (nombre minimum de caractères, unicité...).
- Pour les caractères du mot de passe saisi soient brouillés à l'écran, vous pouvez utiliser la commande CHANGER JEU DE CARACTERES avec la police spéciale %password.

- Des règles d’expiration peuvent être appliquées au moment de la validation de la boîte de dialogue : date d’expiration, changement forcé à la première connexion, verrouillage du compte après plusieurs saisies erronées, mémorisation des mots de passe déjà utilisés...
3. Lorsque la saisie est validée, les informations requises (nom d’utilisateur et mot de passe) sont passées à la commande CHANGER UTILISATEUR COURANT afin d’ouvrir la base avec les privilèges du compte utilisateur.

Exemple

L'exemple suivant affiche la boîte de dialogue de connexion :

```
CHANGER UTILISATEUR COURANT
```

Référence

CHANGER MOT DE PASSE.

ECRIRE ACCES PLUGIN (plugIn; groupe)

Paramètre	Type		Description
plugIn	Entier long	→	Numéro du plug-in
groupe	Alpha	→	Nom du groupe à associer au plug-in

Description

La commande ECRIRE ACCES PLUGIN permet de spécifier par programmation le groupe d'utilisateurs autorisé à utiliser chaque plug-in "sérialisé" installé dans la base. Cette définition permet de gérer la répartition des licences des plug-ins.

Note : Cette opération peut également être effectuée en mode Développement dans l'éditeur de groupes.

Passez dans le paramètre plugIn le numéro du plug-in auquel associer un groupe d'utilisateurs. Les licences de plug-ins incluent les licences Web et SOAP de 4D Client. Vous pouvez passer une des constantes suivantes du thème "Licence disponible" :

Constante	Type	Valeur
Licence 4D Draw	Entier long	808464694
Licence 4D for OCI	Entier long	808465208
Licence 4D View	Entier long	808465207
Licence 4D Write	Entier long	808464697
Licence Web 4D Client	Entier long	808465209
Licence SOAP 4D Client	Entier long	808465465
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D for ADO	Entier long	808465714
Licence 4D for MySQL	Entier long	808465712
Licence 4D for PostgreSQL	Entier long	808465713
Licence 4D for Sybase	Entier long	808465715

Passez dans le paramètre groupe le nom du groupe dont les utilisateurs seront autorisés à utiliser le plug-in.

Note : L'autorisation d'accès à un plug-in n'est accordée qu'à un seul groupe. Si un autre groupe disposait déjà de l'autorisation d'accès, il perd ce privilège à l'issue de l'exécution de la commande.

Référence

Lire acces plugin, LISTE PLUGINS.

Ecrire proprietes groupe (réfGroupe; nom; propriétaire{; membres}) → Numérique

Paramètre	Type		Description
réfGroupe	Numérique	→	Numéro de référence unique du groupe activé ou -1 pour ajouter un groupe de Super_Utilisateur -2 pour ajouter un groupe d'Administrateur
nom	Alpha	→	Nouveau nom de groupe
propriétaire	Numérique	→	Numéro de référence unique de l'utilisateur ou le propriétaire du nouveau groupe
membres	Tableau num	→	Nouveaux membres du groupe
Résultat	Numérique	←	Numéro de référence unique du nouveau groupe

Description

Ecrire proprietes groupe vous permet de modifier et de mettre à jour les propriétés d'un groupe existant dont vous passez le numéro de référence unique dans réfGroupe, ou d'ajouter un nouveau groupe affilié au Super_Utilisateur ou à l'Administrateur.

Si vous modifiez les propriétés d'un groupe existant, vous devez passer son numéro de référence tel que retourné dans la commande LIRE LISTE GROUPE. Les numéros de référence de groupe sont les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15001 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous voulez ajouter un nouveau groupe affilié au Super_Utilisateur, passez -1 dans réfGroupe. Si vous voulez ajouter un nouveau groupe affilié à l'Administrateur, passez -2 dans réfGroupe.

Si le groupe a bien été créé, Ecrire proprietes groupe retourne son numéro de référence unique.

Si vous ne passez pas -1, -2 ou un numéro de référence de groupe valide, Ecrire proprietes groupe ne fait rien et retourne 0.

Avant d'appeler cette routine, vous passez le nouveau nom du groupe et le numéro du propriétaire du groupe dans les paramètres nom et propriétaire. Si vous ne voulez pas modifier toutes les propriétés du groupe (à part ses membres, voir ci-dessous), passez les valeurs retournées par LIRE PROPRIETES GROUPE dans les paramètres que vous voulez laisser inchangés.

Si vous ne passez pas le paramètre optionnel membres, la liste courante des membres du groupe reste inchangée. Si vous le faites lors d'une création d'un groupe, le groupe n'aura pas de membres.

Note : Le propriétaire d'un groupe n'est pas automatiquement défini comme membre du groupe qu'il possède. C'est à vous de l'y inclure explicitement, à l'aide du paramètre membres.

Si vous passez le paramètre optionnel membres, vous modifiez toute la liste des membres pour ce groupe. Avant d'appeler cette routine, vous devez remplir le tableau membres avec les numéros de référence uniques des utilisateurs et/ou des groupes devant appartenir au groupe. Ces numéros peuvent être les suivants :

Numéro de référence du membre	**Description membre**
1	Super_Utilisateur
2	Administrateur
1 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous voulez enlever tous les membres d'un groupe, passez un tableau vide dans le paramètre membres.

Référence

LIRE LISTE GROUPE, LIRE LISTE UTILISATEURS, LIRE PROPRIETES GROUPE.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande Ecrire proprietes groupe ou si le système de mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

Ecrire proprietes utilisateur (réfUtilisateur; nom; démarrage; motDePasse; nbUtilisation; dernièreUtilisation{; adhésions{; groupePropriétaire}}) → Numérique

Paramètre	Type		Description
réfUtilisateur	Numérique	→	Numéro de référence unique du compte de l'utilisateur ou -1 pour l'ajout d'un utilisateur affilié au Super_Utilisateur ou -2 pour l'ajout d'un utilisateur affilié à l'Administrateur
nom	Alpha	→	Nouveau nom de l'utilisateur
démarrage	Alpha	→	Nom de la nouvelle méthode de démarrage
motDePasse	Alpha	→	Nouveau mot de passe (non crypté) ou * pour ne pas modifier le mot de passe
nbUtilisation	Numérique	→	Nouveau nombre d'utilisations de la base
dernièreUtilisation	Date	→	Nouvelle date de dernière utilisation de la base
adhésions	Tableau num	→	Numéros de référence des groupes auxquels l'utilisateur appartient
groupePropriétaire	Numérique	→	Numéro de référence du groupe propriétaire de l'utilisateur
Résultat	Numérique	←	Numéro de référence unique du nouvel utilisateur

Description

Ecrire proprietes utilisateur vous permet de modifier et de mettre à jour les propriétés d'un compte actif d'utilisateur existant dont le numéro de référence est passé dans le paramètre réfUtilisateur, ou d'ajouter un nouvel utilisateur affilié soit au Super_Utilisateur soit à l'Administrateur.

Si vous modifiez les propriétés d'un utilisateur existant, vous devez passer le numéro de référence qui vous est renvoyé par la commande LIRE LISTE UTILISATEURS.

Si le compte d'utilisateur n'existe pas ou a été supprimé, Ecrire proprietes utilisateur retourne 0 et l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR. Sinon, vous pouvez appeler la fonction Utilisateur supprime pour tester le compte de l'utilisateur avant d'appeler Ecrire proprietes utilisateur.

Les numéros de référence pour les utilisateurs peuvent être les suivants :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15000	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Si vous voulez ajouter un nouvel utilisateur affilié au Super_Utilisateur, il faut passer -1 à réfUtilisateur. Si vous voulez ajouter un nouvel utilisateur affilié à l'Administrateur, il faut passer -2 à réfUtilisateur.

Si l'utilisateur a bien été créé ou modifié, Ecrire proprietes utilisateur retourne son numéro de référence unique d'utilisateur.
Si vous ne passez pas un numéro de référence d'utilisateur valide, Ecrire proprietes utilisateur ne fait rien et retourne 0.

Lorsque vous appelez cette commande, vous passez le nouveau nom, la nouvelle méthode de démarrage, le nouveau mot de passe, le nouveau nombre d'utilisations et la nouvelle date de dernière utilisation pour l'utilisateur dans les paramètres nom, démarrage, motDePasse, nbUtilisation et dernièreUtilisation. Vous passez un mot de passe non crypté dans le paramètre motDePasse. 4D cryptera ce mot de passe avant de le sauvegarder dans le compte de l'utilisateur.
Si le nouveau nom d'utilisateur passé dans nom n'est pas unique (un utilisateur de même nom existe déjà), la commande ne fait rien et l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

Si vous ne voulez pas modifier toutes les propriétés de l'utilisateur (à part son groupe, voir ci-dessous), appelez au préalable LIRE PROPRIETES UTILISATEUR et passez les valeurs retournées dans celles que vous ne voulez pas modifier. Si vous ne voulez pas modifier le mot de passe de l'utilisateur, passez * dans le paramètre motDePasse. Cela vous permet de changer les autres propriétés du compte de l'utilisateur, sans changer le mot de passe de ce compte.

Si vous ne passez pas le paramètre optionnel adhésions, les adhésions de l'utilisateur restent inchangées. Si vous ne passez pas ce paramètre en cas d'ajout d'un utilisateur, il ne fera partie d'aucun groupe.

Si vous passez le paramètre optionnel adhésions, vous modifiez toutes les adhésions pour l'utilisateur. Avant d'appeler cette commande, vous devez remplir le tableau adhésions avec les numéros de référence uniques des groupes dont l'utilisateur devra faire partie.

Si vous passez le paramètre facultatif groupePropriétaire, vous indiquez le numéro de référence du groupe “propriétaire” de l’utilisateur, c’est-à-dire le groupe propriétaire par défaut des objets créés par cet utilisateur.

Les numéros de référence pour les groupes peuvent être les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15001 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous voulez annuler les adhésions d'un utilisateur, passez un tableau vide dans le paramètre adhésion.

Référence

LIRE LISTE GROUPE, LIRE LISTE UTILISATEURS, LIRE PROPRIETES UTILISATEUR, SUPPRIMER UTILISATEUR, Utilisateur supprime, Valider mot de passe.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler Ecrire proprietes utilisateur ou si le système de mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

Licence disponible

Licence disponible {(licence)} → Booléen

Paramètre	Type		Description
licence	Numérique	→	Plug-in duquel tester la validité de la licence
Résultat	Booléen	←	Vrai si le plug-in est disponible, sinon Faux

Description

La commande Licence disponible permet de connaître la disponibilité d'un plug-in. Elle est utile, par exemple, pour afficher ou masquer des fonctions nécessitant la présence d'un plug-in.

La commande Licence disponible peut être utilisée de trois manières différentes :

• Le paramètre licence est omis : dans ce cas, la commande retourne Faux si l'application 4D est en mode démonstration.

• Vous passez dans le paramètre licence une des constantes du thème "Licence disponible" :

Constante	Type	Valeur
Licence 4D Draw	Entier long	808464694
Licence 4D for OCI	Entier long	808465208
Licence 4D View	Entier long	808465207
Licence 4D Web	Entier long	808464945
Licence 4D Write	Entier long	808464697
Licence Web 4D Client	Entier long	808465209
Licence SOAP 4D Client	Entier long	808465465
Licence 4D SOAP	Entier long	808465464
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D for ADO	Entier long	808465714
Licence 4D for MySQL	Entier long	808465712
Licence 4D for PostgreSQL	Entier long	808465713
Licence 4D for Sybase	Entier long	808465715

Dans ce cas, la commande retourne Vrai si le plug-in correspondant dispose d'une licence d'utilisation. La commande tient compte des éventuelles attributions de licences effectuées en mode Développement ou via la commande ECRIRE ACCES PLUGIN.
Licence disponible retourne Faux si le plug-in fonctionne en mode démonstration.

• Vous passez directement dans le paramètre licence le numéro d'ID de la ressource "4BNX" du plug-in. Le fonctionnement de la commande est dans ce cas identique à celui décrit ci-dessus.

Référence

ECRIRE ACCES PLUGIN, Lire acces plugin, LISTE PLUGINS.

Lire acces plugin (plugIn) → Alpha

Paramètre	Type		Description
plugIn	Entier long	→	Numéro du plug-in
Résultat	Alpha	←	Nom du groupe associé au plug-in

Description

La commande Lire acces plugin retourne le nom du groupe d'utilisateurs autorisé à utiliser le plug-in dont le numéro a été passé dans le paramètre plugIn. Si aucun groupe n'est associé au plug-in, la commande retourne une chaîne vide ("").

Passez dans le paramètre plugIn le numéro du plug-in duquel vous souhaitez connaître le groupe d'utilisateurs associé. Les licences de plug-ins incluent les licences Web et SOAP de 4D Client. Vous pouvez passer une des constantes suivantes du thème "Licence disponible" :

Constante	Type	Valeur
Licence 4D Draw	Entier long	808464694
Licence 4D for OCI	Entier long	808465208
Licence 4D View	Entier long	808465207
Licence 4D Write	Entier long	808464697
Licence Web 4D Client	Entier long	808465209
Licence SOAP 4D Client	Entier long	808465465
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D for ADO	Entier long	808465714
Licence 4D for MySQL	Entier long	808465712
Licence 4D for PostgreSQL	Entier long	808465713
Licence 4D for Sybase	Entier long	808465715

Référence

LIRE LISTE PLUGIN, ECRIRE ACCES PLUGIN.

LIRE LISTE GROUPE (nomsGroupe; numérosGroupe)

Paramètre	Type		Description
nomsGroupe	Tableau alpha	←	Noms des groupes tels qu'ils apparaissent dans l'éditeur de Mots de passe
numérosGroupe	Tableau num	←	Numéros de référence uniques pour chaque groupe

Description

LIRE LISTE GROUPE remplit les tableaux nomsGroupe et numérosGroupe avec les noms et les numéros de référence uniques des groupes tels qu'ils apparaissent dans l'éditeur de Mots de passe.

Le tableau numérosGroupe, synchronisé avec le tableau nomsGroupe, est rempli avec les numéros de référence uniques des groupes. Ces numéros sont les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Référence

Ecrire proprietes groupe, LIRE LISTE UTILISATEURS, LIRE PROPRIETES GROUPE.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande LIRE LISTE GROUPE ou si le système de Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

LIRE LISTE UTILISATEURS (nomsUtil; réfUtil)

Paramètre	Type		Description
nomsUtil	Tableau alpha	←	Noms des utilisateurs tels qu'ils apparaissent dans l'éditeur de Mots de passe
réfUtil	Tableau num	←	Numéros de référence uniques pour chaque utilisateur

Description

La commande LIRE LISTE UTILISATEURS remplit les tableaux nomsUtil et réfsUtil avec les noms et les numéros de référence uniques des utilisateurs tels qu'ils apparaissent dans la fenêtre des Mots de passe de 4D.

Le tableau nomsUtil est rempli avec les noms des utilisateurs, y compris ceux dont le compte est supprimé (les utilisateurs dont le nom apparaît en vert dans la fenêtre des mots de passe).

Note : Utilisez la commande Utilisateur supprime pour savoir si un compte utilisateur est supprimé.

Le tableau réfsUtil, synchronisé avec nomsUtil, est rempli avec les numéros de référence uniques des utilisateurs. Ces numéros peuvent avoir les valeurs suivantes :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Référence

Ecrire proprietes utilisateur, LIRE LISTE GROUPE, LIRE PROPRIETES UTILISATEUR.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande LIRE LISTE UTILISATEURS ou si le système des Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

LIRE PROPRIETES GROUPE (réfGroupe; nom; propriétaire{; membres})

Paramètre	Type		Description
réfGroupe	Numérique	→	Numéro de référence du groupe
nom	Alpha	←	Nom du groupe
propriétaire	Numérique	←	Numéro de référence du propriétaire du groupe
membres	Tableau num	←	Membres du groupe

Description

LIRE PROPRIETES GROUPE retourne les propriétés du groupe dont le numéro de référence est passé dans réfGroupe. Vous passez le numéro de référence du groupe retourné par la commande LIRE LISTE GROUPE. Les numéros de référence des groupes sont les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Si vous ne passez pas un numéro de référence valide, LIRE PROPRIETES GROUPE renvoie des paramètres vides.

Après l'appel de la commande, vous récupérez le nom et le numéro du propriétaire du groupe dans les paramètres nom et propriétaire.

Si vous passez le paramètre optionnel membres, ce tableau contiendra les numéros de référence uniques des utilisateurs qui appartiennent au groupe. Les numéros de référence des membres de groupe sont les suivants :

Numéro de référence du membre	Description membre
1	Super_Utilisateur
2	Administrateur
1 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15011 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Référence

Ecrire proprietes groupe, LIRE LISTE GROUPE, LIRE LISTE UTILISATEURS.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande LIRE PROPRIETES GROUPE ou si le système de Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

LIRE PROPRIETES UTILISATEUR (réfUtilisateur; nom; démarrage; motDePasse; nbUtilisations; dernièreUtilisation{; adhésions{; groupePropriétaire}})

Paramètre	Type		Description
réfUtilisateur	Numérique	→	Numéro de référence unique de l'utilisateur
nom	Alpha	←	Nom de l'utilisateur
démarrage	Alpha	←	Nom de la méthode de démarrage
motDePasse	Alpha	←	Chaîne vide
nbUtilisations	Numérique	←	Nombre d'utilisations de la base
dernièreUtilisation	Date	←	Date de la dernière utilisation de la base
adhésions	Tableau num	←	Numéros de référence des groupes auxquels l'utilisateur appartient
groupePropriétaire	Numérique	←	Numéro de référence du groupe propriétaire de l'utilisateur

Description

LIRE PROPRIETES UTILISATEUR retourne les informations concernant l'utilisateur dont le numéro de référence est passé dans le paramètre réfUtilisateur. Vous devez passer le numéro de référence retourné par la commande LIRE LISTE UTILISATEURS.

Si le compte d'utilisateur n'existe pas ou a été supprimé, l'erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR. Sinon, vous pouvez appeler la fonction Utilisateur supprime pour tester le compte de l'utilisateur avant d'appeler LIRE PROPRIETES UTILISATEUR.

Les numéros de référence pour les utilisateurs peuvent être les suivants :

Numéro de référence de l'utilisateur	Description utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le second, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le second, et ainsi de suite).

Après l'appel, pour chaque utilisateur dont la référence est réfUtilisateur, vous récupérez aussi le nom, la méthode de démarrage, le nombre d'utilisations et la date de la dernière utilisation de la base dans les paramètres nom, démarrage, nbUtilisation et dernièreUtilisation.

Note : La commande LIRE PROPRIETES UTILISATEUR ne retourne plus le mot de passe crypté dans le paramètre motDePasse. Depuis la version 6.0.2 de 4D, une chaîne vide est toujours retournée dans ce paramètre. Si vous souhaitez contrôler le mot de passe d'un utilisateur, utilisez la fonction Valider mot de passe.

Si vous passez le paramètre facultatif adhésion, vous récupérez le numéro de référence unique du groupe auquel l'utilisateur appartient.

Si vous passez le paramètre facultatif groupePropriétaire, vous récupérez le numéro de référence du groupe “propriétaire” de l’utilisateur, c’est-à-dire le groupe propriétaire par défaut des objets créés par cet utilisateur.

Les numéros de référence pour les groupes peuvent être les suivants :

Numéro de référence du groupe	Description du groupe
15001 à 32767	Groupe créé par le Super_Utilisateur ou le propriétaire du groupe (le groupe n°15001 est le 1er groupe créé par le Super_Utilisateur, le groupe n°15002 est le second, etc.
-15001 à -32768	Groupe créé par l'Administrateur ou le propriétaire du groupe (le groupe n°-15001 est le 1er groupe créé par l'Administrateur, le groupe n°-15002 est le second, etc.

Référence

Ecrire proprietes utilisateur, LIRE LISTE GROUPE, LIRE LISTE UTILISATEURS, Valider mot de passe.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler la commande LIRE PROPRIETES UTILISATEUR ou si le système de Mots de passe est déjà ouvert par un autre process, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par APPELER SUR ERREUR.

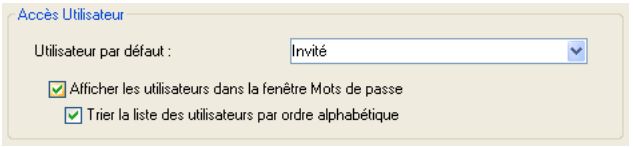
Lire utilisateur par defaut → Numérique

Paramètre	Type	Description
Cette commande ne requiert pas de paramètre		

Résultat	Numérique	← Numéro de référence unique de l'utilisateur

Description

La commande Lire utilisateur par defaut retourne le numéro de référence unique de l'utilisateur désigné comme "Utilisateur par défaut" dans la boîte de dialogue des Préférences de la base :



Les numéros de référence pour les utilisateurs peuvent être les suivants :

Numéro de référence de l'utilisateur	Description de l'utilisateur
1	Super_Utilisateur
2	Administrateur
3 à 15000	Utilisateur créé par le Super_Utilisateur (l'utilisateur n°3 est le 1er utilisateur créé par le Super_Utilisateur, l'utilisateur n°4 est le deuxième, et ainsi de suite).
-11 à -15010	Utilisateur créé par l'Administrateur (l'utilisateur n°-11 est le 1er utilisateur créé par l'Administrateur, l'utilisateur n°-12 est le deuxième, et ainsi de suite).

Si aucun utilisateur par défaut n'est défini, la commande retourne 0.

SUPPRIMER UTILISATEUR (réfUtilisateur)

Paramètre	Type		Description
réfUtilisateur	Numérique	→	Numéro d'identification de l'utilisateur à supprimer

Description

La commande SUPPRIMER UTILISATEUR supprime l'utilisateur dont le numéro est passé dans réfUtilisateur. Vous devez passer un numéro valide d'utilisateur, retourné par la commande LIRE LISTE UTILISATEURS.

Si le compte de l'utilisateur n'existe pas ou a déjà été supprimé, une erreur -9979 est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Seuls le Super_Utilisateur et l'Administrateur peuvent supprimer des utilisateurs. Il n'est pas possible à l'Administrateur de supprimer un utilisateur créé par le Super_Utilisateur.

Les utilisateurs supprimés n'apparaissent plus dans la fenêtre des mots de passe qui est affichée lorsque vous appelez CHANGER UTILISATEUR. Cependant, afin de maintenir des numéros d'utilisateur uniques, le compte de l'utilisateur est conservé dans le système de mots de passe. Les utilisateurs supprimés sont affichés en vert dans l'éditeur des utilisateurs en mode Développement.

Référence

Ecrire proprietes utilisateur, LIRE LISTE UTILISATEURS, LIRE PROPRIETES UTILISATEUR, Utilisateur supprime.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler SUPPRIMER UTILISATEUR ou si un autre process accède déjà au système de mots de passe, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs installée par la commande APPELER SUR ERREUR.

Utilisateur courant → Alpha

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Alpha	←	Nom de l'utilisateur courant

Description

Utilisateur courant retourne le "nom d'utilisateur" de l'utilisateur courant.

Exemple

Reportez-vous à l'exemple de la commande Appartient au groupe.

Référence

Appartient au groupe, CHANGER MOT DE PASSE, CHANGER UTILISATEUR COURANT.

Utilisateur supprime

Utilisateur supprime (réfUtilisateur) → Booléen

Paramètre	Type		Description
réfUtilisateur	Numérique	→	Numéro d'identification de l'utilisateur
Résultat	Booléen	←	Vrai = le compte de l'utilisateur est supprimé ou n'existe pas Faux = le compte de l'utilisateur est actif

Description

La commande Utilisateur supprime teste le compte de l'utilisateur dont le numéro d'identification unique est passé dans réfUtilisateur.

Si le compte n'existe pas ou a été supprimé, la fonction Utilisateur supprime retourne Vrai. Sinon, elle retourne Faux.

Référence

Ecrire proprietes utilisateur, LIRE PROPRIETES UTILISATEUR, SUPPRIMER UTILISATEUR.

Gestion des erreurs

Si vous n'avez pas les privilèges d'accès pour appeler Utilisateur supprime ou si un autre process accède déjà au système de mots de passe, une erreur de privilège d'accès est générée. Vous pouvez intercepter cette erreur avec une méthode de gestion d'erreurs utilisant la commande APPELER SUR ERREUR.

UTILISATEURS VERS BLOB (utilisateurs)

Paramètre	Type		Description
utilisateurs	BLOB	→	BLOB devant contenir les utilisateurs
		←	Comptes utilisateurs (crypté)

Description

La commande UTILISATEURS VERS BLOB stocke dans le BLOB utilisateurs la liste de tous les comptes d'utilisateurs et les groupes de la base créés par l'Administrateur.

Seuls l'Administrateur et le Super_Utilisateur de la base peuvent exécuter cette commande. Si un autre utilisateur tente de l'exécuter, la commande ne fait rien et une erreur de privilège (-9949) est générée.

Le BLOB généré est automatiquement encrypté et ne peut être lu que par la commande BLOB VERS UTILISATEURS. Vous pouvez stocker ce BLOB dans un fichier disque ou dans un champ.
Cette commande équivaut à l'enregistrement des groupes et utilisateurs depuis la fenêtre de gestion des groupes de la Boîte à outils, à la différence près qu'elle permet de stocker les comptes utilisateurs dans un champ BLOB et non uniquement dans un fichier.

Ce principe permet de conserver une sauvegarde des utilisateurs parmi les données de la base, et ainsi de mettre en place un mécanisme de sauvegarde et de chargement automatiques des utilisateurs en cas de mise à jour de la structure de la base (en effet, les informations relatives aux comptes utilisateurs sont stockées par 4D dans le fichier de structure de la base).

Référence

BLOB VERS UTILISATEURS.

Valider mot de passe

Valider mot de passe (réfUtilisateur; motDePasse) → Booléen

Paramètre	Type		Description
réfUtilisateur	Numérique	→	N° de référence unique
motDePasse	Alpha	→	Mot de passe non crypté
Résultat	Booléen	←	Vrai = mot de passe correct Faux = mot de passe incorrect

Description

La commande Valider mot de passe retourne Vrai si la chaîne passée dans motDePasse est le mot de passe du compte utilisateur dont le n° de référence est passé dans réfUtilisateur.

La commande est temporisée afin d'éviter des attaques par "force brute" (essais automatiques de multiples combinaisons de noms d'utilisateurs/mots de passe). Ainsi, au bout du quatrième appel à la commande, elle n'est exécutée qu'après 10 secondes d'attente. Cette temporisation est globale au poste de travail.

Exemple

L'exemple suivant vérifie que "Laurel" est le mot de passe de l'utilisateur "Hardy" :

```
LIRE LISTE UTILISATEURS(atNomUtil;alRefUtil)
$vlElem:=Chercher dans tableau(atNomUtil;"Hardy")
Si ($vlElem>0)
    Si (Valider mot de passe(alRefUtil{$vlElem};"Laurel"))
        ALERTE("Oui !")
    Sinon
        ALERTE("Dommage !")
    Fin de si
Sinon
    ALERTE("Nom d'utilisateur inconnu")
Fin de si
```

Référence

Ecrire proprietes utilisateur, LIRE PROPRIETES UTILISATEUR.

62

Variables

ECRIRE VARIABLES

ECRIRE VARIABLES (doc; variable{; variable2; ...; variableN})

Paramètre	Type		Description
doc	Alpha	→	Nom du document dans lequel sauvegarder la ou les variable(s)
variable	Variable	→	Variable(s) à sauvegarder

Description

La commande ECRIRE VARIABLES sauvegarde une ou plusieurs variable(s) dans un document disque dont le nom est passé dans le paramètre doc.

Les variables ne doivent pas obligatoirement être du même type, mais doivent avoir le type Texte, numérique, Date, Heure, Booléen ou Image.

Si vous passez une chaîne vide ("") dans doc, une boîte de dialogue standard d'enregistrement de fichiers apparaît, permettant à l'utilisateur de donner un nom au document à créer. Dans ce cas, la variable système Document récupère le nom du document, s'il a bien été créé.

Si les variables ont été correctement sauvegardées, la variable système OK prend la valeur 1. Sinon, OK prend la valeur 0.

Note : Lorsque vous écrivez des variables dans des documents à l'aide de la commande ECRIRE VARIABLES, 4D utilise un format de données qui lui est propre. Vous ne pouvez récupérer les variables qu'avec la commande LIRE VARIABLES. N'utilisez pas les commandes RECEVOIR VARIABLE ou RECEVOIR PAQUET pour lire un document créé par ECRIRE VARIABLES.

ATTENTION : La commande ECRIRE VARIABLES ne permet pas de sauvegarder les variables de type Tableau. Pour cela, vous devez utiliser les commandes du thème BLOB.

Exemple

L'exemple suivant enregistre trois variables dans un fichier nommé PrefsUti :

```
ECRIRE VARIABLES ("PrefsUti"; VSNom; VLCode; VGIconPict)
```

Variables et ensembles système

Si l'opération s'est correctement déroulée, la variable OK prend la valeur 1, sinon elle prend la valeur 0.

Référence

BLOB VERS DOCUMENT, BLOB VERS VARIABLE, DOCUMENT VERS BLOB, LIRE VARIABLES, VARIABLE VERS BLOB, Variables système.

EFFACER VARIABLE (variable)

Paramètre	Type		Description
variable	Variable	→	Nom de la variable à effacer

Description

EFFACER VARIABLE réinitialise variable à la valeur par défaut de son type (par exemple chaîne vide pour les types Alpha et Texte, 0 — zéro — pour les variables numériques, aucun élément pour un tableau etc.). La variable existe toujours en mémoire.

La variable passée dans variable doit être une variable process ou interprocess.

Note : Il n'est pas nécessaire d'effacer les variables process à la fin de l'exécution d'un process, 4D s'en charge automatiquement.

Les variables locales, c'est-à-dire celles dont le nom est précédé du symbole dollar ($), ne peuvent être effacées par EFFACER VARIABLE. Toutefois, chaque variable locale est automatiquement effacée à la fin de l'exécution de la méthode dans laquelle elle a été créée.

Exemple

Dans un formulaire, vous utilisez une liste déroulante appelée asMalListeD n'ayant qu'un rôle d'interface utilisateur. Autrement dit, vous exploitez ce tableau lors de la saisie de données, mais une fois que le formulaire est refermé, vous n'en avez plus besoin. Par conséquent, lors de l'événement Sur libération, vous effacez simplement le tableau :

```
    ` Méthode objet liste déroulante asMalListeD
Au cas ou
    : (Evenement formulaire=Sur chargement)
            ` Initialiser le tableau comme vous le souhaitez...
        TABLEAU ALPHA(63;asMalListeD;...)
            ` ...
    :(Evenement formulaire=Sur libération)
            ` Vous n'avez plus besoin du tableau
        EFFACER VARIABLE (asMalListeD)
            ` ...
Fin de cas
```

Référence

Indefinie.

Indefinie (variable) → Booléen

Paramètre	Type		Description
variable	Variable	→	Variable à tester
Résultat	Booléen	←	Vrai = Variable actuellement indéfinie Faux = Variable actuellement définie

Description

Indefinie retourne Vrai si variable n'a pas été définie, et Faux si variable a été définie. Une variable est définie si elle a été créée via une directive de compilation ou si une valeur lui a été assignée. Elle est indéfinie dans les autres cas.

Si la base de données a été compilée, la fonction Indefinie retourne Faux pour toutes les variables.

Exemple

Votre application gère un process lorsqu'une commande de menu d'un module particulier de la base est sélectionnée : si le process est déjà créé, vous le passez au premier plan ; s'il n'est pas créé, vous le démarrez. Pour cela, pour chaque module de votre application, vous gérez une variable interprocess <>PID_... initialisée dans la Méthode base Sur ouverture.

Au cours du développement de la base, vous ajoutez de nouveaux modules. Au lieu de devoir à chaque fois modifier la Méthode base Sur ouverture (pour ajouter l'initialisation de la variable <>PID_... correspondante) puis quitter et réouvrir la base pour tout réinitialiser, vous utilisez la fonction Indefinie pour gérer "à la volée" l'ajout d'un nouveau module :

```
      ` Méthode projet M_AJOUT_CLIENTS

Si (Indefinie(<>PID_AJOUT_CLIENTS))
         ` Prise en compte des étapes de développement intermédiaires
   C_ENTIER LONG(<>PID_AJOUT_CLIENTS)
   <>PID_AJOUT_CLIENTS:=0
Fin de si
```

```
Si (<>PID_AJOUT_CLIENTS=0)
    <>PID_AJOUT_CLIENTS:=Nouveau process("P_AJOUT_CLIENTS";64*1024;
                                                    "P_AJOUT_CLIENTS")
Sinon
    MONTRER PROCESS(<>PID_AJOUT_CLIENTS)
    PASSER AU PREMIER PLAN(<>PID_AJOUT_CLIENTS)
Fin de si
    ` Note: P_AJOUT_CLIENTS, la méthode de gestion des process,
    `fixe <>PID_ADD_CUSTOMERS à zéro lorsqu'elle est terminée.
```

Référence

EFFACER VARIABLE.

LIRE VARIABLES (doc; variable{; variable2; ...; variableN})

Paramètre	Type		Description
doc	Alpha	→	Document contenant la ou les variable(s) à lire
variable	Variable	→	Nom de(s) variable(s) devant recevoir les valeurs

Description

La commande LIRE VARIABLES charge une ou plusieurs variables depuis le document désigné par doc. Ce document doit avoir été créé à l'aide de la commande ECRIRE VARIABLES.

Les variables variable, variable2...variableN sont soit créées, soit réécrites si elles existent déjà.

Si vous passez une chaîne vide ("") dans doc, une boîte de dialogue standard d'ouverture de fichiers apparaît, permettant à l'utilisateur de sélectionner le document à ouvrir. Dans ce cas, la variable système Document contiendra le nom du document choisi.

Dans le cadre de bases compilées, les variables utilisées doivent être du même type que celles chargées du disque.

ATTENTION : Cette commande ne traite pas les variables de type Tableau. Pour cela, vous devez utiliser les commandes du thème BLOB.

Exemple

L'exemple suivant charge trois variables d'un document nommé PrefsUti :

```
LIRE VARIABLES ( "PrefsUti"; VSNom; VLCode; VGIconPict)
```

Variables et ensembles système

La variable système OK prend la valeur 1 si les variables ont été correctement chargées, sinon elle prend la valeur 0.

Référence

BLOB VERS DOCUMENT, BLOB VERS VARIABLE, DOCUMENT VERS BLOB, RECEVOIR VARIABLE, VARIABLE VERS BLOB.

63

Web Services (Client)

4D, à compter de la version 2003, prend en charge les “Web Services”, ce qui signifie que le programme vous permet de publier (partie serveur) et/ou d’utiliser (partie client) des Web Services depuis vos bases de données.
Un Web Service est un ensemble de fonctions publié sur un réseau. Ces fonctions peuvent être invoquées et utilisées par toute application compatible Web Services et connectée au réseau. Les Web Services peuvent effectuer tout type de tâche, comme le suivi d’acheminement de colis chez un transporteur, le commerce électronique, le suivi de valeurs boursières, etc.
Pour plus d’informations sur le concept et le fonctionnement des Web Services, reportez-vous au manuel *Mode Développement*.

La souscription aux Web Services dans 4D s’effectue simplement à l’aide de l’Assistant Web Services. Dans la plupart des cas, cet assistant sera suffisant pour vous permettre d’utiliser des Web Services. Toutefois, si vous souhaitez personnaliser certains mécanismes, vous devez utiliser les commandes SOAP client de 4D.

Cette section décrit les commandes utilisées pour la souscription par 4D à des Web Services externes (**partie client**). Pour plus d'informations sur les commandes utilisées lors de la publication des Web Services (partie serveur), reportez-vous aux Commandes du thème Web Services (Serveur).

Note : Par convention, les libellés “SOAP” et “Web Service” ont été utilisés respectivement dans les noms des commandes (et des constantes) côté serveur et côté client. Ces deux notions désignent la même technologie.

APPELER WEB SERVICE (urlAccès; soapAction; nomMéthode; espaceNommage{; typeComposé{; *}})

Paramètre	Type		Description
urlAccès	Chaîne	→	URL d'accès au Web Service
soapAction	Chaîne	→	Contenu du champ SOAPAction
nomMéthode	Chaîne	→	Nom de la méthode
espaceNommage	Chaîne	→	Espace de nommage (Namespace)
typeComposé	Entier long	→	Configuration de types composés (types simples si omis)
*	*	→	Ne pas fermer la connexion

Description

La commande APPELER WEB SERVICE permet d'invoquer un Web Service en envoyant une requête HTTP. Cette requête contient le message SOAP préalablement construit à l'aide de la commande FIXER PARAMETRE WEB SERVICE.

Tout appel ultérieur à la commande FIXER PARAMETRE WEB SERVICE provoquera la construction d'une nouvelle requête. L'exécution d'une commande APPELER WEB SERVICE efface également tout éventuel résultat de Web Service précédemment appelé et le remplace par le(s) nouveau(x) résultats.

Passez dans urlAccès l'URL complet permettant d'accéder au Web Service (ne confondez pas cet URL avec celui du fichier WSDL, décrivant le Web Service).

• **Accès en mode sécurisé (SSL)** : Si vous souhaitez utiliser le Web Service en mode sécurisé (SSL), passez simplement https:// en tête de l'URL au lieu de http://. Ce paramétrage active automatiquement la connexion en mode sécurisé.

Passez dans le paramètre soapAction le contenu du champ SOAPAction de la requête. Ce champ contient généralement la valeur "NomService#NomMéthode".

Passez dans le paramètre nomMéthode le nom de la méthode distante (appartenant au Web Service) que vous souhaitez exécuter.

Passez dans le paramètre espaceNommage l'espace de nommage XML (*Namespace*) utilisé pour la requête SOAP. Pour plus d'informations sur l'espace de nommage XML, reportez-vous au manuel *Mode Développement* de 4D.

Le paramètre optionnel typeComposé permet d'indiquer la configuration des paramètres Web Service envoyés ou reçus (définis à l'aide des commandes FIXER PARAMETRE WEB SERVICE et LIRE RESULTAT WEB SERVICE). La valeur du paramètre typeComposé dépend du mode de publication du Web Service (DOC ou RPC, cf. manuel *Mode Développement* de 4D) et de ses paramètres.
Vous devez passer dans typeComposé l'une des constantes suivantes, placées dans le thème Web Services (Client) :

Constante	Type	Valeur
Web Service dynamique	Entier long	0 (défaut)
Web Service entrée manuel	Entier long	1
Web Service sortie manuel	Entier long	2
Web Service manuel	Entier long	3

Chaque constante correspond à une "configuration" de Web Services. Une configuration représente une combinaison entre le mode de publication (RPC/DOC) et les types de paramètres entrée/sortie simples ou composés (aussi appelés "complexes") mis en oeuvre.

Note : La caractéristique "entrée" ou "sortie" des paramètres s'évalue du point de vue de la méthode proxy/du Web Service :
• un paramètre "entrée" est une valeur passée à la méthode proxy et donc au Web Service,
• un paramètre "sortie" est retourné par le Web Service et donc par la méthode proxy (généralement via $0).

Le tableau suivant fournit les configurations possibles et les constantes correspondantes :

	Paramètres entrée	
Paramètres sortie	**Simples**	**Composés**
Simples	RPC / Web Service dynamique	RPC / Web Service manuel entrée
Composés	RPC / Web Service manuel sortie	RPC ou DOC / Web Service manuel

Les cinq configurations décrites ci-dessous peuvent donc être mises en oeuvre. Dans tous les cas, 4D se charge de formater la requête SOAP à envoyer au Web Service ainsi que son enveloppe. Il vous appartient de formater le contenu de cette requête suivant la configuration utilisée.

Note : Bien qu'étant des types XML composés, les tableaux de données sont gérés par 4D comme des types simples.

Mode RPC, entrée et sortie simples
Cette configuration est la plus simple à utiliser. Dans ce cas, le paramètre typeComposé contient la constante Web Service dynamique ou est omis.
Les paramètres envoyés et les réponses reçues peuvent être manipulés directement, sans traitement préalable.
Reportez-vous à l'exemple de la commande LIRE RESULTAT WEB SERVICE.

Mode RPC, entrée composée et sortie simple
Dans ce cas, le paramètre typeComposé contient la constante Web Service entrée manuel. Avec cette configuration, vous devez passer “manuellement” au Web Service chaque élément xml source sous la forme d'un BLOB, à l’aide de la commande FIXER PARAMETRE WEB SERVICE.
Il vous appartient de formater le BLOB initial sous forme d’élément xml valide. Ce BLOB doit contenir comme premier élément le premier élément “fils” supposé de l’élément <Body> de la requête finale.

• Exemple

```
C_BLOB($1)
C_BOOLEEN($0)

FIXER PARAMETRE WEB SERVICE("MonBlobXML";$1)
APPELER WEB SERVICE("http://my.domain.com/mon_service";"MonSoapAction";
                    "LaMethode";"http://my.namespace.com/";Web Service entrée manuel)
LIRE RESULTAT WEB SERVICE($0;"MaVarSortie";*)
```

Mode RPC, entrée simple et sortie composée
Dans ce cas, le paramètre typeComposé contient la constante Web Service sortie manuel. Chaque paramètre de sortie sera retourné par le Web Service sous forme d’élément xml stocké dans un BLOB. Vous récupérez ce paramètre à l’aide de la commande LIRE RESULTAT WEB SERVICE. Vous pourrez ensuite analyser le contenu du BLOB reçu à l’aide des commandes XML de 4D.

• Exemple

```
C_BLOB($0)
C_BOOLEEN($1)

FIXER PARAMETRE WEB SERVICE("MaVarEntree";$1)
APPELER WEB SERVICE("http://my.domain.com/mon_service";"MonSoapAction";
                    "LaMethode";"http://my.namespace.com/";Web Service sortie manuel)
LIRE RESULTAT WEB SERVICE($0;"MonXMLSortie";*)
```

Mode RPC, entrée et sortie composées
Dans ce cas, le paramètre typeComposé contient la constante Web Service manuel. Chaque paramètre d’entrée et de sortie devra être stocké sous forme d’élément xml dans des BLOBs, comme décrit dans les deux configurations précédentes.

• Exemple

```
C_BLOB($0)
C_BLOB($1)

FIXER PARAMETRE WEB SERVICE("MonBlobXMLEntree";$1)
APPELER WEB SERVICE("http://my.domain.com/mon_service";"MonSoapAction";
                    "LaMethode";"http://my.namespace.com/";Web Service manuel)
LIRE RESULTAT WEB SERVICE($0;"MonXMLSortie";*)
```

Mode DOC
Une méthode proxy d'appel d'un Web Service DOC est semblable à une méthode proxy d'appel d'un Web Service RPC utilisant des paramètres d'entrée et de sortie composés. La seule différence entre ces deux configurations se situe au niveau du contenu xml des paramètres BLOB passés et reçus. Du point de vue de 4D, la construction et l'envoi de la requête SOAP sont identiques.

• Exemple

```
C_BLOB($0)
C_BLOB($1)

FIXER PARAMETRE WEB SERVICE("MonXMLEntree";$1)
APPELER WEB SERVICE("http://my.domain.com/mon_service";"MonSoapAction";
                    "LaMethode";"http://my.namespace.com/";Web Service manuel)
LIRE RESULTAT WEB SERVICE($0;"MonXMLSortie";*)
```

Note : Dans le cas des Web Services DOC, la valeur des chaînes (ci-dessus "MonXMLEntree" et "MonXMLSortie") passées en paramètres n'a pas d'importance ; il est même possible de passer des chaînes vides (""). En effet, ces valeurs ne sont pas utilisées dans la requête SOAP contenant le document xml. Il est toutefois obligatoire de passer ces paramètres.

Pour utiliser un Web Service publié en mode DOC (ou en mode RPC avec types composés), il est conseillé de procéder de la manière suivante :

• Générer la méthode proxy à l'aide de l'Assistant Client Web Services.
La méthode proxy sera appelée de la manière suivante :
$BlobXMLresult:=$proxy_MethodeDOC($BlobXMLparam)

• Prendre connaissance du contenu des requêtes SOAP à envoyer au Web Service à l'aide d'un outil de test en ligne (par exemple http://soapclient.com/soaptest.html). Ce type d'outil permet, à partir du WSDL du Web Service, de générer des formulaires HTML de test.

• Copier le contenu xml généré à partir du premier fils de <body>.

• Ecrire la méthode permettant de placer les valeurs réelles des paramètres dans le code xml ; ce code doit ensuite être placé dans le BLOB $BlobXMLparam.

• Pour l'analyse de la réponse, vous pouvez également utiliser un outil de test en ligne, ou tirer parti du WSDL qui spécifie les éléments retournés.

Le paramètre * permet d'optimiser les appels. Lorsqu'il est passé, la commande ne referme pas la connexion utilisée par le process à l'issue de son exécution. Dans ce cas, l'appel suivant à APPELER WEB SERVICE réutilise cette même connexion si le paramètre * est passé, et ainsi de suite. Pour refermer la connexion, il suffit d'exécuter la commande APPELER WEB SERVICE sans le paramètre *. Ce mécanisme permet d'accélérer sensiblement les traitements en cas d'appels successifs de plusieurs Web Services sur le même serveur, notamment en configuration WAN (via Internet par exemple). A noter que ce mécanisme s'appuie sur le paramétrage "keep-alive" du serveur Web. Ce paramétrage définit généralement un nombre maximal de requêtes via une même connexion, et peut même les interdire. Si les requêtes APPELER WEB SERVICE enchaînées dans la même connexion atteignent ce nombre maximal ou si les connexions keep-alive ne sont pas autorisées, 4D créera une nouvelle connexion pour chaque requête.

Référence

FIXER PARAMETRE WEB SERVICE, LIRE RESULTAT WEB SERVICE.

Variables et ensembles système

Si la requête est correctement acheminée et que le Web Service l'a acceptée, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0 et une erreur est retournée.

AUTHENTIFIER WEB SERVICE (nom; motDePasse{; méthodeAuth}{; *})

Paramètre	Type		Description
nom	Chaîne	→	Nom de l'utilisateur
motDePasse	Chaîne	→	Mot de passe de l'utilisateur
méthodeAuth	Entier long	→	Méthode d'authentification : 0 ou omis=non définie, 1=BASIC, 2=DIGEST
*	*	→	Si passé : authentification par proxy

Description

La commande AUTHENTIFIER WEB SERVICE vous permet d'utiliser des Web Services nécessitant l'authentification de l'application cliente. Les méthodes BASIC et DIGEST sont prises en charge ainsi que la présence d'un proxy.

Note : Pour plus d'informations sur les méthodes d'authentification BASIC et DIGEST, reportez-vous à la section Sécurité des connexions.

Passez dans les paramètres nom et motDePasse les informations d'identification requises (nom d'utilisateur et mot de passe). Ces informations seront encodées et ajoutées à la requête HTTP envoyée au Web Service via la commande APPELER WEB SERVICE. Il est donc nécessaire d'appeler la commande AUTHENTIFIER WEB SERVICE avant la commande APPELER WEB SERVICE.

Le paramètre facultatif méthodeAuth permet d'indiquer la méthode d'authentification à utiliser pour le prochain appel de la commande APPELER WEB SERVICE. Vous pouvez passer l'une des valeurs suivantes :
• 2 = utiliser la méthode d'authentification DIGEST
• 1 = utiliser la méthode d'authentification BASIC
• 0 (ou paramètre omis) = utiliser la méthode appropriée. Dans ce cas, 4D envoie une requête supplémentaire afin de négocier la méthode d'authentification.

Si vous passez le paramètre *, vous indiquez que les informations d'authentification s'adressent à un proxy HTTP. Ce paramétrage doit être mis en oeuvre lorsqu'il existe un proxy nécessitant une authentification entre le client du Web Service et le Web Service lui-même. Si le Web Service est lui-même authentifié, une double authentification est requise (cf. exemple).

Par défaut, les informations d'authentification sont réinitialisées après chaque requête. Vous devez donc utiliser la commande AUTHENTIFIER WEB SERVICE avant chaque APPELER WEB SERVICE. Il est toutefois possible de conserver temporairement ces informations à l'aide d'une option de la commande FIXER OPTION WEB SERVICE. Dans ce cas, il n'est pas nécessaire d'exécuter la commande AUTHENTIFIER WEB SERVICE avant chaque APPELER WEB SERVICE.

En cas d'échec de l'authentification, le serveur SOAP retourne une erreur que vous pouvez identifier à l'aide de la commande Lire infos erreur Web Service.

Exemple

Authentification auprès d'un Web Service situé derrière un proxy :

```
    `Authentification au Web Service en mode DIGEST
AUTHENTIFIER WEB SERVICE("SoapUser";"123";2)
    `Authentification au proxy en mode par défaut
AUTHENTIFIER WEB SERVICE("ProxyUser";"456";*)
APPELER WEB SERVICE(...)
```

Référence

APPELER WEB SERVICE, Lire infos erreur Web Service.

FIXER OPTION WEB SERVICE (option; valeur)

Paramètre	Type		Description
option	Entier long	→	Code de l'option à fixer
valeur	Entier long | Texte	→	Valeur de l'option

Note préliminaire : Cette commande est destinée aux utilisateurs avancés des Web Services. Son emploi est facultatif.

Description

La commande FIXER OPTION WEB SERVICE permet de définir différentes options qui seront utilisées lors de la prochaine requête SOAP déclenchée par la commande APPELER WEB SERVICE.
Vous pouvez appeler cette commande autant de fois qu'il y a d'options à fixer.

Passez dans le paramètre option le numéro de l'option à définir et dans le paramètre valeur la nouvelle valeur de l'option. Vous pouvez utiliser pour ces deux paramètres une des constantes prédéfinies suivantes, situées dans le thème "Web Services (Client)" :

Constante (param *option*)	Type	Valeur
Web Service timeout HTTP	Entier long	1
Web Service header SOAP	Entier long	2
Web Service version SOAP	Entier long	3
Web Service afficher dial auth	Entier long	4
Web Service effacer infos auth	Entier long	5
Web Service compression HTTP	Entier long	6

Constante (param *valeur*)	Type	Valeur
Web Service SOAP_1_1	Entier long	0
Web Service SOAP_1_2	Entier long	1
Web Service compression deflate	Entier long	1

Voici la description des options et des valeurs possibles :

• option = Web Service timeout HTTP
valeur = "timeout" de la partie cliente exprimé en secondes.
Le timeout de la partie cliente est le délai d'attente du client Web Service en cas de non-réponse du serveur. A l'issue de ce délai, le client referme la session, la requête est perdue. Par défaut, ce délai est de 180 secondes. Il peut être modifié en raison de caractéristiques particulières (état du réseau, spécificités du Web Service, etc.).

• option = Web Service header SOAP
valeur = référence d'élément xml racine à insérer en tant que header (en-tête) de la requête SOAP.
Cette option permet d'insérer un header dans la requête SOAP générée par la commande APPELER WEB SERVICE. Par défaut, les requêtes SOAP ne comportent pas d'en-tête spécifique. Cependant, certains Web Services requièrent la présence de cet en-tête, par exemple pour la gestion de paramètres d'identification.

• option = Web Service version SOAP
valeur = Web Service SOAP_1_1 ou Web Service SOAP_1_2
Cette option permet de préciser la version du protocole SOAP utilisée dans la requête. Passez dans valeur la constante Web Service SOAP_1_1 pour indiquer la version 1.1 et la constante Web Service SOAP_1_2 pour indiquer la version 1.2.

• option = Web Service afficher dial auth
valeur = 0 (ne pas afficher le dialogue) ou 1 (afficher le dialogue)
Cette option gère l'affichage de boîte de dialogue d'authentification lors de l'exécution de la commande APPELER WEB SERVICE. Par défaut, cette commande ne provoque jamais l'affichage de la boîte de dialogue, vous devez en principe utiliser la commande AUTHENTIFIER WEB SERVICE. Toutefois, si vous souhaitez qu'une boîte de dialogue d'authentification apparaisse pour que l'utilisateur saisisse ses identifiants, vous devez utiliser cette option : passez 1 dans valeur pour afficher la boîte de dialogue, et 0 sinon. La boîte de dialogue n'apparaît que si le service Web requiert une authentification.

• option = Web Service effacer infos auth
valeur = 0 (ne pas effacer les informations) ou 1 (les effacer)
Cette option permet d'indiquer à 4D de mémoriser les informations d'authentification de l'utilisateur (nom d'utilisateur, mot de passe, méthode, etc.), dans le but de les réutiliser par la suite. Par défaut, ces informations sont effacées après chaque exécution de la commande APPELER WEB SERVICE. Passez 0 dans valeur pour les mémoriser et 1 pour les effacer. A noter que lorsque vous passez 0, les informations sont conservées pendant la session mais ne sont pas stockées..

• option = Web Service compression HTTP
valeur = Web Service compression deflate
Cette option permet d'activer un mécanisme interne de compression des requêtes SOAP afin d'accélérer les échanges inter-applications 4D. Lorsque vous exécutez l'instruction FIXER OPTION WEB SERVICE(Web Service compression HTTP; Web Service compression deflate) sur le client 4D du Web Service, les données de la prochaine requête SOAP envoyée par le client seront compressées en utilisant un mécanisme standard HTTP avant leur envoi au serveur SOAP 4D. Le serveur décompressera et analysera la requête puis répondra en utilisant automatiquement le même mécanisme. Seule la requête suivant l'appel de la commande FIXER OPTION WEB SERVICE est affectée. Vous devez donc appeler cette commande chaque fois que vous voulez utiliser la compression. Par défaut, 4D ne compresse pas les requêtes HTTP des Web Services.

Notes :
• "Deflate" est le nom de l'algorithme de compression utilisé en interne par 4D.
• Ce mécanisme ne peut pas être utilisé pour des requêtes adressées à un serveur SOAP 4D d'une version antérieure à la 11.3.
• Afin de vous permettre d'optimiser encore ce fonctionnement, des options supplémentaires configurent le seuil et le taux de compression des requêtes. Ces options sont accessibles via la commande FIXER PARAMETRE BASE.

L'ordre d'appel des options n'a pas d'importance. Si une même option est définie plusieurs fois, seule la valeur du dernier appel est prise en compte.

Exemples

(1) Insertion d'un en-tête personnalisé dans la requête SOAP :

```
    ` Création d'une référence XML
C_ALPHA(16;vRefRacine;vRefElement)
vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath)
    `Modification de l'en-tête SOAP avec la référence
FIXER OPTION WEB SERVICE(Web Service header SOAP;vRefElement)
```

(2) Utilisation de la version 1.2 du protocole SOAP :

```
FIXER OPTION WEB SERVICE(Web Service version SOAP;Web Service SOAP_1_2)
```

Référence

APPELER WEB SERVICE.

FIXER PARAMETRE WEB SERVICE (nom; valeur{; typeSOAP})

Paramètre	Type		Description
nom	Chaîne	→	Nom du paramètre à inclure dans la requête SOAP
valeur	Variable	→	Variable 4D contenant la valeur du paramètre
typeSOAP	Chaîne	→	Type SOAP du paramètre

Description

La commande FIXER PARAMETRE WEB SERVICE permet de définir un paramètre utilisé pour une requête SOAP cliente. Appelez autant de fois cette commande qu'il y a de paramètres dans la requête.

Passez dans nom le nom du paramètre tel qu'il doit apparaître dans la requête SOAP.

Passez dans valeur la variable 4D contenant la valeur du paramètre. Dans le cadre des méthodes proxy, cette variable est généralement $1, $2, $3, etc., correspondant à un paramètre 4D passé à la méthode proxy lors de son appel. Il est toutefois possible d'utiliser des variables intermédiaires.

Note : Chaque variable ou tableau 4D utilisé(e) doit être préalablement déclaré(e) à l'aide des commandes des thèmes "Compilateur" et "Tableaux".

Par défaut, 4D détermine automatiquement le type SOAP le plus approprié pour le paramètre nom en fonction du contenu de valeur. L'indication du type est incluse dans la requête.
Toutefois, vous pouvez vouloir "forcer" la définition du type SOAP du paramètre. Dans ce cas, vous pouvez passer le paramètre optionnel typeSOAP ; vous devez utiliser une des chaînes de caractères suivantes (types de données primaires) :

typeSOAP	Description
string	Chaîne
int	Entier long
boolean	Booléen
float	Réel 32 bits
decimal	Réel avec décimale
double	Réel 64 bits
duration	Durée en années mois jours heures minutes secondes, par exemple : P1Y2M3DT10H30M
datetime	Date et heure au format ISO8601, par exemple 2003-05-31T13:20:00
time	Heure, par exemple 13:20:00

date	Date, par exemple 2003-05-31
gyearmonth	Année et mois (calendrier grégorien), par exemple 2003-05
gyear	Année (calendrier grégorien), par exemple 2003
gmonthday	Mois et jour (calendrier grégorien), par exemple --05-31
gday	Jour (calendrier grégorien), par exemple ---31
gmonth	Mois (calendrier grégorien), par exemple --10--
hexbinary	Valeur exprimée en hexadécimal
base64binary	BLOB
anyuri	Uniform Resource Identifier (URI), par exemple : http://www.societe.com/page
qname	Nom XML qualifié (espace de nommage et partie locale)
notation	Attribut Notation

Note : Pour plus d'informations sur les types de données XML, reportez-vous à l'URL http://www.w3.org/TR/xmlschema-2/

Exemple

Cet exemple définit deux paramètres :

```
C_TEXTE($1)
C_TEXTE($2)
FIXER PARAMETRE WEB SERVICE("ville";$1)
FIXER PARAMETRE WEB SERVICE("pays";$2)
```

Référence

APPELER WEB SERVICE, LIRE RESULTAT WEB SERVICE.

Lire infos erreur Web Service (typeInfo) → Chaîne

Paramètre	Type		Description
typeInfo	Entier long	→	Information à récupérer
Résultat	Chaîne	←	Information sur la dernière erreur SOAP

Description

La commande Lire infos erreur Web Service retourne des informations relatives à l'erreur éventuellement générée lors de l'exécution de la dernière requête SOAP adressée à un Web Service. Cette commande doit généralement être appelée dans le cadre d'une méthode d'appel sur erreur installée par la commande APPELER SUR ERREUR.

Le paramètre typeInfo vous permet d'indiquer le type d'information que vous souhaitez obtenir. Vous devez passer une des constantes suivantes, placées dans le thème Web Services (Client) :

Constante	Type	Valeur
Web Service code erreur	Entier long	0
Web Service message	Entier long	1
Web Service code erreur HTTP	Entier long	2
Web Service origine erreur	Entier long	3

Ces constantes pemettent de récupérer les valeurs suivantes :

• Web Service code erreur : Code d'erreur principal (défini par 4D). Ce code est également retourné dans la variable système *Error*.
Voici la liste des codes pouvant être retournés :
9910 : Erreur Web Service (voir aussi Web Service origine erreur)
9911 : Erreur de l'analyseur xml
9912 : Erreur HTTP (voir aussi Web Service code erreur HTTP)
9913 : Erreur réseau
9914 : Erreur interne

• Web Service message : Message détaillé décrivant l'erreur. Le type de message diffère suivant le type d'erreur principale.
- Si erreur principale = 9910 (Erreur Web Service) : la cause de l'erreur SOAP est retournée (ex : "méthode appelée inexistante").
- Si erreur principale = 9911 (Erreur de l'analyseur xml) : l'emplacement de l'erreur dans le document xml est retourné.

- Si erreur principale = 9912 (Erreur HTTP) :
    - si l'erreur HTTP est située dans l'intervalle [300-400] (problèmes lié à l'emplacement du document demandé), le nouvel emplacement de l'URL demandé est retourné.
    - pour tout autre code d'erreur HTTP, le <body> est renvoyé.
- Si erreur principale = 9913 (Erreur réseau) : la cause de l'erreur réseau est retournée (ex : "AdresseServeur : erreur DNS")
- Si erreur principale = 9914 (Erreur interne) : la cause de l'erreur interne est retournée

• Web Service code erreur HTTP : Code de l'erreur HTTP (à utiliser en cas d'erreur principale 9912).

• Web Service origine erreur : Cause de l'erreur (retournée par le protocole SOAP — à utiliser en cas d'erreur principale 9910).
    - Version Mismatch (les versions ne correspondent pas).
    - Must Understand (un paramètre défini comme obligatoire n'a pas pu être interprété par le serveur)
    - Client Fault (erreur client)
    - Server Fault (erreur serveur)
    - Encoding Unknown (encodage inconnu)

Une chaîne vide est retournée lorsqu'aucune information n'est disponible, en particulier si la dernière requête SOAP n'a pas généré d'erreur.

LIRE RESULTAT WEB SERVICE (valeurRetour{; nomRetour{; *}})

Paramètre	Type		Description
valeurRetour	Variable	←	Valeur retournée par le Web Service
nomRetour	Chaîne	→	Nom du paramètre à récupérer
*		→	Libérer la mémoire

Description

La commande LIRE RESULTAT WEB SERVICE permet de récupérer une valeur renvoyée par le Web Service à l'issue du traitement effectué.

Note : Cette commande doit être utilisée uniquement après la commande APPELER WEB SERVICE.

Le paramètre valeurRetour reçoit la valeur renvoyée par le Web Service. Passez dans ce paramètre une variable 4D. Cette variable est généralement $0, correspondant à la valeur retournée par la méthode proxy. Il est toutefois possible d'utiliser des variables intermédiaires (vous devez utiliser des variables process uniquement).

Note : Chaque variable ou tableau 4D utilisé(e) doit être préalablement déclaré(e) à l'aide des commandes des thèmes "Compilateur" et "Tableaux".

Le paramètre optionnel nomRetour permet de spécifier le nom du paramètre à récupérer. Toutefois, comme la plupart des Web Services retournent une seule valeur, ce paramètre n'est généralement pas nécessaire.

Le paramètre *, optionnel, indique au programme de libérer la mémoire consacrée au traitement de la requête. Vous devez passer ce paramètre après la récupération de la dernière valeur renvoyée par le Web Service.

Exemple

Imaginons un Web Service retournant l'heure courante dans n'importe quelle ville du monde. Les paramètres reçus par le Web Service sont le nom de la ville et le code du pays. Le Web Service retourne alors l'heure correspondante. La méthode proxy d'appel pourrait être de la forme suivante :

```
C_TEXTE($1)
C_TEXTE($2)
C_HEURE($0)
```

```
FIXER PARAMETRE WEB SERVICE("ville";$1)
FIXER PARAMETRE WEB SERVICE("code_pays";$2)

APPELER WEB SERVICE("http://www.villesdumonde.com/WS";"WSHeures#Heure_ville";
                    "Heure_ville";"http://www.villesdumonde.com/namespace/default")

Si (OK=1)
   LIRE RESULTAT WEB SERVICE($0;"retour";*)
Fin de si
```

Référence

APPELER WEB SERVICE, FIXER PARAMETRE WEB SERVICE.

64

Web Services (Serveur)

La publication de Web Services dans 4D s'effectue simplement à l'aide d'options dans les propriétés des méthodes. Dans la plupart des cas, ce fonctionnement sera suffisant pour vous permettre de publier des Web Services. Toutefois, si vous souhaitez personnaliser certains mécanismes, utiliser des tableaux de données..., vous devez utiliser les commandes SOAP serveur de 4D.

Cette section décrit les commandes utilisées lors de la publication dans 4D de Web Services (**partie serveur**). Pour des informations générales sur les Web Services ou la description des commandes utilisées lors de la souscription aux Web Services (partie client), reportez-vous à la section Commandes du thème Web Services (Client).

Note : Les libellés "SOAP" et "Web Service" ont été utilisés respectivement dans les noms des commandes (et des constantes) côté serveur et côté client par convention uniquement. Ces deux notions désignent la même technologie.

DECLARATION SOAP (variable; type; entrée_sortie{; alias})

Paramètre	Type		Description
variable	Variable 4D	→	Variable référençant un argument SOAP entrant ou sortant
type	Entier long	→	Type 4D vers lequel pointe l'argument
entrée_sortie	Entier long	→	1 = Entrée SOAP, 2 = Sortie SOAP
alias	Chaîne	→	Nom publié pour cet argument lors des échanges SOAP

Description

La commande DECLARATION SOAP permet de déclarer explicitement le type des paramètres utilisés dans une méthode 4D publiée comme Web Service.

Lorsqu'une méthode est publiée en tant que Web Service, les paramètres standard $0, $1... $n sont utilisés pour décrire les paramètres du Web Service auprès du monde extérieur (notamment dans le fichier WSDL). Le protocole SOAP exigeant que les paramètres soient explicitement nommés, 4D utilise par défaut les noms "FourD_arg0, FourD_arg1 ... FourD_argn".

Ce fonctionnement par défaut peut toutefois s'avérer problématique pour les raisons suivantes :
• Il n'est pas possible de déclarer $0 ou $1, $2, etc. en tant que tableau. Il est donc nécessaire d'utiliser des pointeurs, mais dans ce cas il reste à connaître le type des valeurs pour la génération du fichier WSDL.
• Il peut être utile ou nécessaire de personnaliser les noms des paramètres (entrants et sortants).
• Vous pouvez vouloir utiliser des paramètres tels que des structures XML et des références DOM.
• Il peut également être nécessaire de retourner des valeurs d'une taille supérieure à 32 Ko (limite des arguments de type Texte dans les bases de données en mode non Unicode).
• Enfin, ce fonctionnement rend impossible le retour de plus d'une valeur par appel (dans $0).

La commande DECLARATION SOAP permet de s'affranchir de ces limites. Vous pouvez exécuter cette commande pour chaque paramètre entrant et sortant et lui assigner un nom et un type.

Note : Même si la commande DECLARATION SOAP est utilisée, il est nécessaire de déclarer les variables et tableaux 4D dans la méthode *Compiler_Web* à l'aide les commandes du thème "Compilateur".

Passez dans variable la variable 4D à référencer dans l'appel du Web Service.
Attention, vous pouvez référencer uniquement des variables process ou les arguments des méthodes 4D ($0 à $n). Les variables locales et interprocess ne peuvent pas être utilisées.

Par défaut, seuls des arguments de type Texte pouvant être utilisés, les réponses du serveur SOAP sont en principe limitées à 32 Ko dans les bases de données en mode non Unicode. Il est toutefois possible de retourner des arguments SOAP d'une taille supérieure à 32 Ko, en utilisant des BLOBs. Pour cela, il suffit de déclarer les arguments en type BLOB avant d'appeler la commande DECLARATION SOAP (cf. exemple 4).

Note : Côté client, si vous souscrivez à ce type de Web Service avec 4D, l'assistant Web Services générera naturellement une variable de type Texte dans la méthode proxy. Pour pouvoir l'utiliser, il vous suffit de retyper cette variable de retour en BLOB dans la méthode proxy.

Passez dans type le type 4D correspondant. La plupart des types de variables et de tableaux 4D peuvent être employés. Vous pouvez utiliser les constantes ci-dessous, placées dans le thème "Types champs et variables", ainsi que, pour les types XML, deux constantes du thème "Web Services (Serveur)" :

Constantes "Types champs et variables"	**Type**	**Valeur**
Est un BLOB	Entier long	30
Est un booléen	Entier long	6
Est un entier	Entier long	8
Est un entier long	Entier long	9
Est un numérique	Entier long	1
Est un tableau booléen	Entier long	22
Est un tableau chaîne	Entier long	21
Est un tableau date	Entier long	17
Est un tableau entier	Entier long	15
Est un tableau entierlong	Entier long	16
Est un tableau numérique	Entier long	14
Est un tableau texte	Entier long	18
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11
Est une variable chaîne	Entier long	24

Constantes "Web Services (Serveur)"	**Type**	**Valeur**
Est un XML	Entier long	36
Est une référence DOM	Entier long	37

Passez dans entrée_sortie une valeur indiquant si le paramètre traité est “entrant” (c’est-à-dire correspondant à une valeur reçue par la méthode) ou “sortant” (c’est-à-dire correspondant à une valeur retournée par la méthode). Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème “Web Services (Serveur)” :

Constante	Type	Valeur
SOAP entrée	Entier long	1
SOAP sortie	Entier long	2

Utilisation de types XML : Vous pouvez déclarer des variables de type "structure XML" et "référence DOM", aussi bien en entrée qu’en sortie, via les constantes Est un XML et Est une référence DOM. Lorsque des paramètres de ce type sont définis, aucun traitement ni encodage ne leur est appliqué, les données sont transmises telles quelles (cf. exemple 5).
• *Paramètres sortants* :
- Est un XML indique que le paramètre contient une structure XML,
- Est une référence DOM indique que le paramètre contient la référence DOM d’une structure XML. Dans ce cas, l’insertion de la structure XML dans le message SOAP équivaut à l’exécution de la commande DOM EXPORTER VERS VARIABLE.
Note : Dans le cas de références DOM utilisées en paramètres sortants, il est recommandé d’utiliser des références globales, créées par exemple au démarrage et closes à la fermeture de l’application. En effet, une référence DOM créée au sein du Web Service lui-même ne peut pas être refermée avec DOM FERMER XML sinon le Web Service ne retourne plus rien. Les appels multiples au Web Service impliquent alors la création d’autant de références DOM non refermées, ce qui peut provoquer une saturation de la mémoire.
• *Paramètres entrants* :
- Est un XML indique que le paramètre doit recevoir un argument XML envoyé par le client SOAP.
- Est une référence DOM indique que le paramètre doit recevoir la référence DOM d’une structure XML correspondant à l’argument XML envoyé par le client SOAP.
• *Modification de la WSDL* : Les structures XML seront déclarées par 4D du type "anyType" (indéterminé) dans la WSDL. Si vous souhaitez typer précisément une structure XML, vous devez sauvegarder le fichier WSDL et ajouter manuellement le schéma de données souhaité dans la section <types> de la WSDL.

Méthode COMPILER_WEB : Les arguments SOAP entrants référencés à l’aide de variables 4D (et non via les arguments des méthodes 4D) doivent être préalablement déclarés dans la méthode projet *COMPILER_WEB*. En effet, l’utilisation de variables process dans les méthodes Web Services nécessite leur déclaration avant l’appel de la méthode. La méthode projet *COMPILER_WEB* est appelée, si elle existe, à chaque requête SOAP acceptée. Par défaut, la méthode *COMPILER_WEB* n’existe pas. Vous devez la créer explicitement.
A noter que la méthode *COMPILER_WEB* est également appelée par le serveur Web de 4D lors de la réception de requêtes Web “classiques” de type POST (cf. section URLs et actions de formulaires).

Passez dans alias le nom de l'argument tel qu'il doit apparaître dans la WSDL et dans les échanges SOAP. **Attention**, ce nom doit être unique dans l'appel RPC (paramètres entrants et sortants confondus), sinon seule la dernière déclaration sera prise en compte par 4D.

Note : Les noms des arguments ne doivent pas débuter par un chiffre ni contenir d'espaces. En outre, pour éviter tout risque d'incompatibilité, il est recommandé de ne pas utiliser de caractères étendus (tels que des caractères accentués).

Si le paramètre alias est omis, 4D utilisera par défaut le nom de la variable ou FourD_argN pour les arguments des méthodes 4D ($0 à $n).

Note : La commande DECLARATION SOAP doit être incluse dans la méthode publiée comme Web Service. Il n'est pas possible de l'appeler d'une autre méthode.

Exemples

(1) Cet exemple spécifie un nom de paramètre :

```
    ` Dans la méthode COMPILER_WEB
C_ENTIER LONG($1)

    `Dans la méthode du service Web
DECLARATION SOAP($1;Est un entier long;SOAP Entrée;"zipcode")
    ` Lors de la génération du fichier WSDL et des appels SOAP, le libellé
    ` zipcode sera utilisé au lieu de fourD_arg1
```

(2) Cet exemple permet de récupérer un tableau de codes postaux sous forme d'entiers longs :

```
    `Dans la méthode COMPILER_WEB
TABLEAU ENTIER LONG(tab_codes;0)

    `Dans la méthode du service Web
DECLARATION SOAP(tab_codes;Est un tableau entierlong;SOAP Entrée;"in_tab_codes")
```

(3) Cet exemple permet de référencer deux valeurs de retour sans spécifier de nom d'argument :

```
DECLARATION SOAP(ret1;Est un entier long;SOAP Sortie)
DECLARATION SOAP(ret2;Est un entier long;SOAP Sortie)
```

(4) Cet exemple permet au serveur SOAP de 4D de retourner un argument d'une taille supérieure à 32 Ko dans les bases de données en mode non Unicode :

```
C_BLOB($0)
DECLARATION SOAP($0; Est un texte ; SOAP sortie)
```

Notez le type Est un texte (et non Est un BLOB). Cette astuce permet un formatage correct de l'argument.

(5) Cet exemple illustre l'effet des différents types de déclarations :

```
TOUT SELECTIONNER([Contact])

`Construction d'une structure XML à partir de la sélection de Contacts et stockage du
`XML dans un BLOB
C_BLOB(ws_vx_xmlBlob)
getContactsXML (->ws_vx_xmlBlob)
    `Récupération de la structure XML dans une variable texte
C_TEXTE(ws_vt_xml)
ws_vt_xml:=BLOB vers texte(ws_vx_xmlBlob;UTF8 Texte sans longueur)
    `Récupération d'une référence DOM vers la structure XML
C_TEXTE(ws_vt_refXML)
ws_vt_refXML:=DOM Analyser variable XML(ws_vt_xml)

    `Test des différentes déclarations
DECLARATION SOAP(ws_vx_xmlBlob;Est un BLOB ;SOAP Sortie ;"contactListsX")
    `Le XML est converti en Base64 par 4D

DECLARATION SOAP(ws_vt_xml;Est un Texte;SOAP Sortie;"contactListsText")
    `Le XML est converti en texte par 4D (< > deviennent des entités)

DECLARATION SOAP(ws_vt_xml;Est un XML;SOAP Sortie;"contactsXML")
    `Le XML est passé en texte XML

DECLARATION SOAP(ws_vx_xmlBlob;Est un XML;SOAP Sortie;"contactsBlob")
    `Le XML est passé en BLOB XML

DECLARATION SOAP(ws_vt_refXML;Est une référence DOM;SOAP Sortie;
                                                              "contactByRef")
    `Le XML est passé en tant que référence
```

Référence

ENVOYER ERREUR SOAP, Fichier donnees verrouille, Lire informations SOAP.

Constantes

Thèmes Types champs et variables et Web Services (Serveur).

ENVOYER ERREUR SOAP (typeErreur; description)

Paramètre	Type		Description
typeErreur	Entier long	→	1 = Erreur Client, 2 = Erreur Serveur
description	Chaîne	→	Description de l'erreur à envoyer au client SOAP

Description

La commande ENVOYER ERREUR SOAP permet de retourner une erreur à un client SOAP en indiquant l'origine de l'erreur : client ou serveur. Utiliser cette commande vous permet de signaler une erreur à un client sans devoir retourner de résultat.

Par exemple, une erreur côté client peut être détectée lorsque vous publiez un Web Service "Racine_carree" et qu'un client envoie une requête avec un nombre négatif ; vous pouvez utiliser cette commande afin d'indiquer au client qu'une valeur positive est requise.

Une erreur possible côté serveur peut être par exemple un manque de mémoire survenu lors de l'exécution de la méthode.

Passez dans typeErreur l'origine de l'erreur. Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème "Web Services (Serveur)" :

Constante	Type	Valeur
SOAP erreur client	Entier long	1
SOAP erreur serveur	Entier long	2

Passez dans description un descriptif de l'erreur. Si l'implémentation du client est conforme, l'erreur pourra être traitée.

Exemple

Pour reprendre l'exemple du Web Service "Racine_carree" fourni dans la description de la commande, l'instruction suivante peut être utilisée pour traiter les requêtes sur des nombres négatifs :

```
ENVOYER ERREUR SOAP(SOAP erreur client;"Valeurs positives requises")
```

Référence

DECLARATION SOAP, Lire informations SOAP.

Est une requete SOAP → Booléen

Paramètre	Type		Description
Cette commande ne requiert pas de paramètre			
Résultat	Booléen	←	Vrai si la requête est SOAP, Faux sinon

Description

La commande Est une requete SOAP retourne Vrai si le code en cours d'exécution fait partie d'une requête SOAP.

Cette commande peut être utilisée pour des raisons de sécurité dans la Méthode base Sur authentification Web afin de déterminer la nature des requêtes reçues.

Référence

DECLARATION SOAP, Méthode base Sur authentification Web.

Lire informations SOAP

Lire informations SOAP (numInfo) → Chaîne

Paramètre	Type		Description
numInfo	Entier long	→	Numéro du type d'information SOAP à lire
Résultat	Chaîne	←	Information SOAP

Description

La commande Lire informations SOAP permet de récupérer sous forme de chaîne de caractères différents types d'informations concernant une requête SOAP.

Lorsque vous traitez une requête SOAP, il peut être utile d'obtenir des informations supplémentaires — en-dehors des valeurs des paramètres RPC — sur la requête. Par exemple, pour des raisons de sécurité, vous pouvez utiliser cette commande dans la Méthode base Sur authentification Web afin de connaître le nom de la méthode Web Service demandée.

Passez dans le paramètre numInfo le numéro du type d'information SOAP à connaître. Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème "Web Services (Serveur)" :

Constante	Type	Valeur
SOAP nom méthode	Entier long	1
SOAP nom service	Entier long	2

• SOAP nom méthode = nom de la méthode offerte comme Web Service sur le point d'être exécutée.
• SOAP nom service = nom du Web Service auquel appartient la méthode.

Note : Pour des raisons de sécurité également, il est possible de définir la taille maximale des requêtes Web Services adressées à 4D. Ce paramétrage est effectué à l'aide de la commande FIXER PARAMETRE BASE (thème "Définition structure").

Référence

ENVOYER ERREUR SOAP, FIXER PARAMETRE BASE.

65

XML DOM

4D inclut un ensemble de commandes permettant d'écrire et d'analyser des objets contenant des données XML (*eXtensible Markup Language*).

A propos du langage XML
Le langage XML est une norme d'échange de données. Il est basé sur l'emploi de balises permettant de décrire de manière précise les données échangées ainsi que leur structure. Les fichiers XML sont des fichiers au format Texte, leur contenu est analysé (*parsing*) par les applications qui importent les données. Aujourd'hui, de nombreuses applications prennent en charge ce format.
Pour plus d'informations sur le XML, reportez-vous, par exemple, au site http://xmlfr.org.

Pour la prise en charge du XML, 4D utilise une librairie nommée Xerces.dll développée par la société Apache Foundation. 4D prend en charge XML version 1.0.

Note : 4D permet également d'importer et d'exporter directement des données au format XML à l'aide de l'éditeur d'import/export standard.

DOM et SAX
Les commandes de ce thème sont préfixées DOM. En effet, 4D propose deux ensembles distincts de commandes XML: les commandes DOM (Document Object Model) et SAX (Simple API XML), qui constituent deux modes d'analyse différents des documents XML.
• Le mode DOM effectue l'analyse d'une source XML et construit sa structure (son "arbre") en mémoire. De ce fait, l'accès à chaque élément de la source est extrêmement rapide. Cependant, la totalité de l'arbre étant contenu dans la mémoire, le traitement de gros documents XML peut dépasser la capacité de la mémoire et provoquer des erreurs.
• Le mode SAX ne construit pas d'arbre en mémoire. Dans ce mode, des "événements" (tels que le début et la fin d'un élément) sont générés lors de l'analyse de la source. Ce mode autorise l'analyse de documents XML de toute taille, quelle que soit la quantité de mémoire disponible. Les commandes SAX sont regroupées dans le thème "XML SAX". Pour plus d'informations, reportez-vous à la section Présentation des commandes XML SAX.
Pour plus d'informations sur les standards XML, vous pouvez consulter les sites http://www.saxproject.org/?selected=event et http://www.w3schools.com/xml/.

Création, ouverture et fermeture des documents XML via DOM
Les objets créés, modifiés ou analysés par les commandes DOM de 4D peuvent être des textes, des URLs, des documents ou des BLOBs. Les commandes DOM utilisées pour l'ouverture des objets XML dans 4D sont DOM Analyser source XML et DOM Analyser variable XML.
De nombreuses commandes permettent ensuite de lire, d'analyser et d'écrire les éléments et les attributs. La récupération des erreurs s'effectue via la commande LIRE ERREUR XML (commune aux deux standards XML).
La commande DOM FERMER XML permet de finalement refermer la source.

Utilisation de la notation XPath
Trois commandes XML DOM (DOM Creer element XML, DOM Chercher element XML et DOM ECRIRE VALEUR ELEMENT XML) acceptent la notation XPath pour l'accès aux éléments XML.
La *notation XPath* est issue du *langage XPath*, consacré à la navigation à l'intérieur des structures XML. Elle permet de désigner directement des éléments au sein d'une structure XML via une syntaxe du type "chemin d'accès", sans devoir nécessairement indiquer le chemin complet pour y parvenir. Soit par exemple la structure suivante :

```
<RootElement>
    <Elem1>
        <Elem2>
            <Elem3 Font=Verdana  Size=10> </Elem3>
        </Elem2>
    </Elem1>
</RootElement>
```

La notation XPath permet d'accéder à l'élément 3 via la syntaxe /RootElement/Elem1/Elem2/Elem3.

4D accepte également les éléments XPath indexés, avec la syntaxe Elément[NumElément]. Soit par exemple la structure suivante :

```
<RootElement>
    <Elem1>
        <Elem2>aaa</Elem2>
        <Elem2>bbb</Elem2>
        <Elem2>ccc</Elem2>
    </Elem1>
</RootElement>
```

La notation XPath permet d'accéder à la valeur "ccc" via la syntaxe /RootElement/Elem1/Elem2[3].

Pour une illustration de la notation XPath, reportez-vous aux exemples des commandes DOM Creer element XML et DOM Chercher element XML.

Jeux de caractères
Les jeux de caractères suivants sont pris en charge par les commandes XML DOM et XML SAX de 4D :
- ASCII
- UTF-8
- UTF-16 (Big/Small Endian)
- UCS4 (Big/Small Endian)
- EBCDIC code pages IBM037, IBM1047 and IBM1140 encodings,
- ISO-8859-1 (ou Latin1)
- Windows-1252.

Terminologie
Le langage XML utilise de nombreux termes et acronymes spécifiques. Cette liste non exhaustive explicite les principales notions XML utilisées par les commandes et fonctions de 4D.

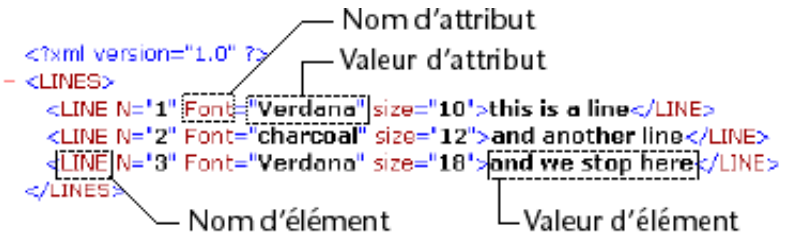
Attribut : Sous-balise XML associée à un élément. Un attribut comporte toujours un nom et une valeur (cf. schéma ci-dessous).

Bien formé : Un document XML est déclaré “bien formé” par l’analyseur XML lorsqu’il est conforme aux spécifications XML génériques. Voir aussi *Validation*.

DTD : *Document Type Declaration* (Déclaration de type de document). La DTD recense l’ensemble des règles et des propriétés spécifiques que doit suivre un document XML. Ces règles définissent notamment le nom et le contenu de chaque balise ainsi que leur contexte. Cette formalisation des éléments permet de vérifier qu’un document XML est conforme (dans ce cas il est déclaré “valide”).
La DTD peut être incluse dans le document XML (DTD interne) ou dans un document tiers (DTD externe). A noter que la DTD n’est pas obligatoire.

Elément : Balise XML. Un élément comporte toujours un nom et une valeur. Facultativement, un élément peut contenir des attributs (cf. schéma).

Eléments et attributs



Enfant : Dans une structure XML, élément d’un niveau directement inférieur à un autre.

Frère : Dans une structure XML, élément du même niveau qu'un autre.

Parent : Dans une structure XML, élément d’un niveau directement supérieur à un autre.

Parsing, **parser** (**Analyser**, **analyseur**) : Action d’analyser le contenu d’un objet structuré afin d’en extraire les informations utiles. Les commandes du thème “XML” permettent d’analyser le contenu de tout objet XML.

Racine (**Root**) : Elément situé au premier niveau d’une structure XML.

RefElément : Référence XML utilisée par les commandes XML de 4D pour désigner une structure XML (documents ou élément). Cette référence est constituée de 8 caractères codés sous forme hexadécimale, ce qui signifie qu'elle est composée de 16 caractères.

Structure XML : objet XML structuré. Cet objet peut être un document, une variable, un élément.

Validation : Un document XML est “validé” par l’analyseur XML lorsqu’il est “bien formé” et conforme aux spécifications de la DTD. Voir aussi *Bien formé*.

XML : *eXtensible Markup Language* (Langage balisé évolutif). Norme d’échange de données informatisées permettant de transférer des données ainsi que leur structure. Le langage XML est basé sur l’emploi de balises et d’une syntaxe spécifiques, à l’instar du langage HTML. Toutefois, à la différence de ce dernier, le langage XML permet de définir des balises personnalisées.

XSL : *eXtensible Stylesheet Language* (Langage des feuilles de style évolutif). Langage permettant de définir des feuilles de style utilisables pour traiter et afficher le contenu d’un document XSL.

DOM Analyser source XML

version 11 (Modifiée)

DOM Analyser source XML (document{; validation{; dtd | schéma}}) → Chaîne

Paramètre	Type		Description
document	Chaîne	→	Chemin d'accès du document
validation	Booléen	→	Vrai = Validation, Faux = Pas de validation
dtd | schéma	Chaîne	→	Emplacement de la DTD ou du schéma XML
Résultat	Chaîne	←	Référence de l'élément XML (16 caractères)

Description

La commande DOM Analyser source XML analyse un document contenant une structure XML et retourne une référence pour ce document. La commande peut valider ou non le document via une DTD ou un schéma XML (document XSD, *XML Schema Definition*). Le document peut être situé sur disque ou sur Internet/Intranet.

Vous pouvez passer dans le paramètre document :
• soit un chemin d'accès complet standard (du type C:\\Dossier\\Fichier\\... sous Windows et MacintoshHD:Dossier:Fichier sous Mac OS),
• soit un chemin Unix sous Mac OS (débutant obligatoirement par /).
• soit un chemin réseau du type http://www.site.com/Fichier ou ftp://public.ftp.com...

Le paramètre booléen validation vous permet d'indiquer si vous souhaitez que la structure soit validée ou non.
• Si validation vaut Vrai, la structure sera validée. Dans ce cas, l'analyseur tentera de valider la structure XML du document sur la base de la référence DTD ou XSD incluse dans le document, ou via la DTD ou le schéma XML désigné(e) par le troisième paramètre s'il est passé.
• Si validation vaut Faux, la structure ne sera pas validée.
Si vous passez Vrai dans validation et omettez le troisième paramètre, la commande tentera de valider la structure XML via une référence DTD ou XSD trouvée dans la structure elle-même. La validation peut être indirecte : si la structure contient une référence vers une fichier DTD qui lui-même contient une référence vers un fichier XSD, la commande tentera d'effectuer les deux validations.

Le troisième paramètre vous permet de désigner une DTD spécifique ou un schéma XML pour l'analyse du document. Si vous utilisez ce paramètre, la commande ne tient pas compte de la DTD référencée dans le document XML.

Validation par DTD
Il existe deux moyens pour désigner une DTD :
• en tant que *référence*. Il vous suffit pour cela de passer le chemin d'accès complet de la nouvelle DTD (extension "dtd") dans le paramètre dtd. Si le document désigné ne contient pas de DTD valide, le paramètre dtd est ignoré et une erreur est générée.
• directement dans un *texte*. Dans ce cas, si le contenu du paramètre débute par "<?xml", 4D considérera qu'il s'agit de la DTD ; dans le cas contraire, 4D considérera qu'il s'agit d'un chemin d'accès.

Validation par schema
Pour valider le document via un schéma XML, il suffit de passer dans le troisième paramètre un fichier ou un URL d'extension"xsd" au lieu de "dtd". La validation par schéma XML est considérée comme plus souple et plus puissante que la validation par DTD. Le langage des documents XSD est basé sur le langage XML. Les schémas XML prennent notamment en charge des types de données. Pour plus d'informations sur les schémas XML, reportez-vous à l'adresse http://www.w3.org/XML/Schema.

Si la validation ne peut être effectuée (pas de DTD ou d'XSD, URL incorrect, etc.), une erreur est générée. La variable système *Error* indique le numéro de l'erreur. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

La commande retourne une chaîne de 16 caractères (*RefElément*) constituant la référence en mémoire de la structure virtuelle du document. Cette référence devra être utilisée avec les autres commandes d'analyse XML.

Important : Une fois que vous n'en avez plus besoin, n'oubliez pas d'appeler la commande DOM FERMER XML avec cette référence afin de libérer la mémoire.

Exemples
(1) Ouverture sans validation d'un document XML situé sur disque :

```
$ref_XML_Struct:=DOM Analyser source XML("C:\\import.xml")
```

(2) Ouverture sans validation d'un document XML situé à côté du fichier de structure de la base :

```
$ref_XML_Struct:=DOM Analyser source XML("import.xml")
```

(3) Ouverture d'un document XML situé sur disque et validation à l'aide d'une DTD située sur le disque :

```
$ref_XML_Struct:=DOM Analyser source XML("C:\\import.xml";Vrai;
                                                "C:\\import_dtd.xml")
```

(4) Ouverture sans validation d'un document XML situé à un URL spécifique :

```
$ref_XML_Struct:=DOM Analyser source XML("http://www.4D.fr/xml/import.xml")
```

Référence

DOM Analyser variable XML, DOM FERMER XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Analyser variable XML

DOM Analyser variable XML (variable{; validation{; dtd}}) → Chaîne

Paramètre	Type		Description
variable	BLOB | Texte	→	Nom de la variable
validation	Booléen	→	Vrai = Validation par le DTD, Faux = Pas de validation
dtd	Chaîne	→	Emplacement de la DTD
Résultat	Chaîne	←	Référence de l'élément XML (16 caractères)

Description

La commande DOM Analyser variable XML analyse une variable de type BLOB ou Texte contenant une structure XML et retourne une référence pour cette variable. La commande peut valider ou non le document.

Passez dans le paramètre variable le nom de la variable BLOB ou Texte contenant l'objet XML.

Le paramètre booléen validation vous permet d'indiquer si vous souhaitez que la structure soit validée ou non à l'aide de la DTD.
• Si validation vaut Vrai, la structure sera validée. Dans ce cas, l'analyseur tentera de valider la structure XML de la variable sur la base de la DTD définie ou référencée dans la variable, ou encore désignée par le paramètre dtd.
• Si validation vaut Faux, la structure ne sera pas validée.

Le troisième paramètre, dtd, vous permet de désigner la DTD spécifique pour l'analyse du document. Si vous utilisez ce paramètre, la commande ne tient pas compte de la DTD référencée dans la variable XML.

Il existe deux moyens pour désigner une DTD :
• en tant que *référence*. Il vous suffit pour cela de passer le chemin d'accès complet de la nouvelle DTD dans le paramètre dtd. Si le document désigné ne contient pas de DTD valide, le paramètre dtd est ignoré et une erreur est générée.
• directement dans un *texte*. Dans ce cas, si le contenu du paramètre débute par "<?xml", 4D considérera qu'il s'agit de la DTD ; dans le cas contraire, 4D considérera qu'il s'agit d'un chemin d'accès.

Si la validation ne peut être effectuée (pas de DTD, URL vers le DTD incorrecte, etc.), une erreur est générée. La variable système *Error* indique le numéro de l'erreur. Vous pouvez intercepter cette erreur à l'aide d'une méthode installée par la commande APPELER SUR ERREUR.

La commande retourne une chaîne de 16 caractères (*RefElément*) constituant la référence en mémoire de la structure virtuelle de la variable. Cette référence devra être utilisée avec les autres commandes d'analyse XML.

Important : Une fois que vous n'en avez plus besoin, n'oubliez pas d'appeler la commande DOM FERMER XML avec cette référence afin de libérer la mémoire.

Exemples

(1) Ouverture sans validation d'un objet XML situé dans une variable Texte 4D :

```
C_TEXTE(maVarTexte)
C_HEURE(vDoc)
C_ALPHA(16;$ref_XML_Struct)

vDoc:=Ouvrir document ("Document.xml")
Si (OK=1)
    RECEVOIR PAQUET(vDoc;maVarTexte;32000)
    FERMER DOCUMENT(vDoc)
    $ref_XML_Struct:=DOM Analyser variable XML(maVarTexte)
Fin de si
```

(2) Ouverture sans validation d'un document XML situé dans un BLOB 4D :

```
C_BLOB(maVarBlob)
C_ALPHA(16;$ref_XML_Struct)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Struct:=DOM Analyser variable XML(maVarBlob)
```

Référence

DOM Analyser source XML, DOM FERMER XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Chercher element XML (refElément; xChemin{; tabRefEléments}) → refElément

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
xChemin	Texte | BLOB	→	Chemin XPath de l'élément à chercher
tabRefEléments	Tab Chaîne 16	←	Liste des références d'éléments trouvés (le cas échéant)
Résultat	refElément	←	Référence de l'élément trouvé (le cas échéant)

Description

La commande DOM Chercher element XML vous permet de rechercher des éléments XML spécifiques dans une structure XML. La recherche débute à l'élément désigné par le paramètre refElément.

Le noeud XML à chercher est défini par le paramètre xChemin, exprimé en notation XPath (cf. paragraphe "Utilisation de la notation XPath" dans la section Présentation des commandes XML DOM). Il est possible d'utiliser des éléments indexés.

Note : Conformément à la norme XML, la recherche différencie les majuscules et les minuscules.

La commande retourne en résultat la référence XML de l'élément trouvé.

Lorsque le tableau chaîne tabRefEléments est passé, la commande le remplit avec la liste des références XML trouvées. Dans ce cas, la commande retourne en résultat le premier élément du tableau tabRefEléments. Ce paramètre est utile lorsque plusieurs éléments de même nom existent à l'emplacement désigné par le paramètre xChemin.

Exemples

(1) Cet exemple permet de rechercher rapidement un élément XML et d'afficher sa valeur :

```
vTrouvé:=DOM Chercher element XML(vRefElem;"Items/Book[15]/Title")
DOM LIRE VALEUR ELEMENT XML(vTrouvé;valeur)
ALERTE("La valeur de l'élément est : \""+valeur+"\"")
```

La même recherche peut également être effectuée ainsi :

```
vTrouvé:=DOM Chercher element XML(vRefElem;"Items//Book[15]")
vTrouvé:=DOM Chercher element XML(vTrouvé;"Book/Title")
DOM LIRE VALEUR ELEMENT XML(vTrouvé;valeur)
ALERTE("La valeur de l'élément est : \""+valeur+"\"")
```

Note : Comme vous pouvez le constater dans cet exemple, le chemin XPath doit toujours débuter par le nom de l'élément courant. Cette précision est importante lorsque vous manipulez des chemins XPath relatifs.

(2) Soit la structure XML suivante :

```
<Racine>
        <Elem1>
            <Elem2>aaa</Elem2>
            <Elem2>bbb</Elem2>
            <Elem2>ccc</Elem2>
        </Elem1>
</Racine>
```

Le code suivant permet de récupérer la référence de chaque élément *Elem2* dans le tableau *tAtrouvés* :

```
TABLEAU ALPHA(16;tAtrouvés;0)
vTrouvé:=DOM Chercher element XML(vRefElem;"/Racine/Elem1/Elem2";tAtrouvés)
```

Référence

DOM Compter elements XML, DOM Creer element XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :
• la référence de l'élément n'est pas valide
• le chemin xPath passé n'est pas valide.

DOM Chercher element XML par ID

DOM Chercher element XML par ID (refElément; id) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
id	Chaîne	→	Valeur de l'attribut ID de l'élément à chercher
Résultat	Chaîne	←	Référence de l'élément trouvé (le cas échéant)

Description

La commande DOM Chercher element XML par ID vous permet de rechercher un élément XML sur la base de la valeur de son attribut ID. La recherche débute à l'élément désigné par le paramètre refElément.

L'attribut ID permet d'associer un identifiant unique à chaque élément. La valeur de l'attribut ID doit être un nom XML valide et doit être unique. Passez dans id la valeur de l'attribut à rechercher.

La commande retourne en résultat la référence XML de l'élément trouvé.

Référence

DOM Chercher element XML.

DOM Compter attributs XML

version 2004 (Modifiée)

DOM Compter attributs XML (refElément) → Entier long

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
Résultat	Entier long	←	Nombre d'attributs

Description

La commande DOM Compter attributs XML retourne le nombre d'attributs XML présents dans l'élément XML désigné par refElément. Pour plus d'informations sur les attributs XML, reportez-vous à la section Présentation des commandes XML DOM.

Exemple

Avant de récupérer les valeurs des éléments dans un tableau, vous souhaitez connaître le nombre d'attributs dans l'élément XML suivant :

```
<?xml version="1.0" ?>
- <LINES>
    <LINE N="1" Font="Verdana" size="10">this is a line</LINE>
    <LINE N="2" Font="charcoal" size="12">and another line</LINE>
    <LINE N="3" Font="Verdana" size="18">and we stop here</LINE>
  </LINES>
```

```
C_BLOB(maVarBlob)
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Enfant)
C_TEXTE(monRésultat)
C_ENTIER LONG($nbAttributs)

$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)

$nbAttributs:=DOM Compter attributs XML($ref_XML_Enfant)
TABLEAU TEXTE(tAttrib;$nbAttributs)
TABLEAU TEXTE(tValAttrib;$nbAttributs)
Boucle($i;1;$nbAttributs)
    DOM LIRE ATTRIBUT XML PAR INDEX($ref_XML_Enfant;$i;tAttrib{$i};tValAttrib{$i})
Fin de boucle
```

Dans l'exemple ci-dessus, *$nbAttributs* vaut 3, *tAttrib{1}* contient "Font", *tAttrib{2}* contient "N" et *tAttrib{3}* contient "size". *tValAttrib* contient, "Verdana", "1" et "10".

Note : Le numéro d'indice ne correspond pas à l'emplacement de l'attribut dans le fichier XML affiché sous forme de texte. En XML, l'indice d'un attribut indique sa position parmi les attributs classés par ordre alphabétique (en fonction de leur nom).

Référence

DOM Compter elements XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Compter elements XML

version 2004 (Modifiée)

DOM Compter elements XML (refElément; nomElément) → Entier long

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElément	Chaîne	→	Nom d'éléments XML à compter
Résultat	Entier long	←	Nombre d'éléments

Description

La commande DOM Compter elements XML retourne le nombre d'éléments "enfants" dépendants de l'élément parent refElément et nommés nomElément.

Référence

DOM Lire element XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Creer element XML (refElément; xChemin{; nomAttribut; valeurAttribut}{; nomAttribut2; valeurAttribut2; ...; nomAttributN; valeurAttributN}) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML racine
xChemin	Texte	→	Chemin XPath de l'élément XML à créer
nomAttribut	Chaîne	→	Attribut à définir
valeurAttribut	Chaîne | Valeur	→	Nouvelle valeur d'attribut
Résultat	Chaîne	←	Référence de l'élément XML créé

Description

La commande DOM Creer element XML permet de créer un nouvel élément dans l'élément XML refElément à l'emplacement du noeud désigné par le paramètre xChemin et de lui ajouter éventuellement des attributs.

Passez dans refElément la référence de l'élément racine (créé par exemple à l'aide de la commande DOM Creer ref XML).

Passez dans xChemin le chemin d'accès de l'élément à créer en notation XPath (cf. paragraphe "Utilisation de la notation XPath" dans la section Présentation des commandes XML DOM). Si des éléments du chemin n'existent pas, ils sont créés.
Il est possible de passer directement dans xChemin un nom d'élément simple afin de créer un sous-élément à partir de l'élément courant (cf. exemple 3).

Note : Si vous avez défini un ou plusieurs espace(s) de nommage pour l'arbre désigné par refElément (cf. commande DOM Creer ref XML), vous devez préfixer le paramètre xChemin du nom de l'espace à utiliser (par exemple "MonNameSpace:MonElément").

Vous pouvez passer dans les paramètres facultatifs nomAttribut et valeurAttribut un couple attribut / valeur d'attribut (sous forme de variables, champs ou valeurs littérales). Vous pouvez passer autant de couples que vous voulez.
Le paramètre valeurAttribut peut être de type texte ou d'un autre type (booléen, entier, réel, heure ou date).

Si vous passez une valeur d'un type autre que texte, 4D se charge de la conversion en texte, selon les principes suivants :

Type	Exemple de valeur convertie
Booléen	"true" ou "false" (non traduit)
Entier	"123456"
Réel	"12.34" (le séparateur décimal est toujours ".")
Heure	"5233" (nombre de secondes)
Date	"2006-12-04T00:00:00Z" (norme RFC 3339)

La commande retourne en résultat la référence XML de l'élément créé.

Exemples

(1) Nous souhaitons créer l'élément suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
        <Elem1>
            <Elem2>
                <Elem3> </Elem3>
                <Elem3> </Elem3>
            </Elem2>
        </Elem1>
</RootElement>
```

Pour cela, il suffit d'écrire :

```
C_ALPHA(16;vRefRacine;vRefElement)
vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath)
vxPath:="/RootElement/Elem1/Elem2/Elem3[2]"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath)
```

(2) Nous souhaitons créer l'élément suivant (comportant des attributs) :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<RootElement>
        <Elem1>
            <Elem2>
                <Elem3 Font=Verdana  Size=10> </Elem3>
            </Elem2>
        </Elem1>
</RootElement>
```

Pour cela, il suffit d'écrire :

```
C_ALPHA(16;vRefRacine;vRefElement)
C_ALPHA(80;$aAttrNom1;$aAttrNom2;$aAttrVal1;$aAttrVal2)
$aAttrNom1:="Font"
$aAttrNom2:="Size"
$aAttrVal1:="Verdana"
$aAttrVal2:="10"

vRefRacine:=DOM Creer ref XML("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElement:=DOM Creer element XML(vRefRacine;vxPath;$aAttrNom1;$aAttrVal1;
                                                    $aAttrNom2;$aAttrVal2)
```

(3) Nous souhaitons créer et exporter la structure suivante :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Racine>
    <Elem1>Hello</Elem1>
</Racine>
```

Nous souhaitons utiliser la syntaxe basée sur un nom d'élément simple. Pour cela, il suffit d'écrire :

```
C_ALPHA(16;$root)
C_ALPHA(16;$ref1)

$root:=DOM Creer ref XML("Racine")
$ref1:=DOM Creer element XML($root;"Elem1")
DOM ECRIRE VALEUR ELEMENT XML($ref1;"Hello")
DOM EXPORTER VERS FICHIER($root;"mondoc.xml")
DOM FERMER XML($root)
```

Référence

DOM Lire element XML, DOM SUPPRIMER ELEMENT XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :
• la référence de l'élément racine n'est pas valide
• le nom de l'élément à créer n'est pas valide (par exemple, s'il débute par un chiffre).

DOM Creer ref XML (racine{; nameSpace}{; nSNom; nSValeur}{; nSNoms; nSValeurs; ...; nSNomN; nSValeurN}) → Chaîne

Paramètre	Type		Description
racine	Chaîne	→	Nom de l'élément racine
nameSpace	Chaîne	→	Valeur de l'espace de nommage (Namespace)
nSNom	Chaîne	→	Nom d'espace de nommage
nSValeur	Chaîne	→	Valeur d'espace de nommage
Résultat	Chaîne	←	Référence de l'élément XML racine

Description

La commande DOM Creer ref XML crée un arbre XML vide en mémoire et retourne sa référence.

Passez dans le paramètre racine le nom de l'élément racine de l'arbre XML.

Passez dans le paramètre facultatif nameSpace la déclaration de la valeur de l'espace de nommage (*namespace*) de l'arbre (par exemple "http://www.4d.com").
Dans ce cas, vous devez préfixer le paramètre racine avec le nom de l'espace de nommage, suivi de : (par exemple "MonNameSpace:MaRacine").

Note : L'espace de nommage (namespace) est une chaîne de caractères permettant de garantir l'unicité des noms de variables XML. En général, un URL du type http://www.monsite.com/monurl est utilisé. Il n'est pas nécessaire que l'URL soit valide sur le site, il faut juste qu'il soit unique.

Vous pouvez déclarer un ou plusieurs espace(s) de nommage supplémentaire(s) dans l'arbre XML généré, à l'aide de couples nSNom / nSValeur. Vous pouvez passer autant de couples nom / valeur d'espace de nommage que vous voulez.

Important : N'oubliez pas d'appeler la commande DOM FERMER XML afin de libérer la mémoire lorsque vous avez terminé d'utiliser l'arbre XML.

Exemples

(1) Création d'un arbre XML simple :

```
C_ALPHA (16;vRefElem)
vRefElem:=DOM Creer ref XML("MaRacine")
```

Ce code produit le résultat suivant :
<**?xml version="1.0" encoding="UTF-8" standalone="no" ?**>
<**MaRacine**/>

(2) Création d'un arbre XML avec un espace de nommage :

```
C_ALPHA (16;vRefElem)
$Racine:="MonNameSpace:MaRacine"
$Namespace:="http://www.4D.com/tech/namespace"
vRefElem:=DOM Creer ref XML($Racine;$Namespace)
```

Ce code produit le résultat suivant :
<**MonNameSpace:MaRacine**
xmlns:MonNameSpace="http://www.4D.com/tech/namespace"/>

(3) Création d'un arbre XML avec plusieurs espaces de nommage :

```
C_ALPHA (16;vRefElem)
C_ALPHA (80;$aNSNom1;$aNSNom2;$aNSValeur1;$aNSValeur2)
$Racine:="MonNameSpace:MaRacine"
$Namespace:="http://www.4D.com/tech/namespace"
$aNSNom1:="NSNom1"
$aNSNom2:= "NSNom2"
$aNSValeur1:="http://www.4D.com/Prod/namespace"
$aNSValeur2:="http://www.4D.com/Mkt/namespace"
vRefElem:=DOM Creer ref
XML($Racine;$Namespace;$aNSNom1;$aNSValeur1;$aNSNom2;$aNSValeur2)
```

Ce code produit le résultat suivant :
<**MonNameSpace:MaRacine**
xmlns:MonNameSpace="http://www.4D.com/tech/nameSpace"
NSNom1="http://www.4D.com/Prod/namespace"
NSNom2="http://www.4D.com/Mkt/namespace"/>

Référence

DOM ECRIRE OPTIONS XML, DOM FERMER XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

DOM ECRIRE ATTRIBUT XML (refElément; nomAttribut; valeurAttribut{; nomAttribut2; valeurAttribut2; ...; nomAttributN; valeurAttributN})

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomAttribut	Chaîne	→	Attribut à définir
valeurAttribut	Chaîne | Valeur	→	Nouvelle valeur d'attribut

Description

La commande DOM ECRIRE ATTRIBUT XML permet d'ajouter un ou plusieurs attribut(s) à l'élément XML dont la référence est passée dans le paramètre refElément. Elle permet également de définir la valeur de chaque attribut défini.

Passez dans les paramètres nomAttribut et valeurAttribut respectivement l'attribut à écrire et sa valeur (sous forme de variables, champs ou valeurs littérales). Vous pouvez passer autant de couples attribut/valeur que vous voulez.
Le paramètre valeurAttribut peut être de type texte ou d'un autre type (booléen, entier, réel, heure ou date). Si vous passez une valeur d'un type autre que texte, 4D se charge de la conversion en texte, selon les principes suivants :

Type	Exemple de valeur convertie
Booléen	"true" ou "false" (non traduit)
Entier	"123456"
Réel	"12.34" (le séparateur décimal est toujours ".")
Heure	"5233" (nombre de secondes)
Date	"2006-12-04T00:00:00Z" (norme RFC 3339)

Exemple

Soit la source XML suivante :

```
<Book>
        <Title>The Best Seller</Title>
</Book>
```

Si le code suivant est exécuté :

```
vAttrName:="Font"
vAttrVal:="Verdana"
DOM ECRIRE ATTRIBUT XML(vRefElem;vAttrName;vAttrVal)
```

Nous obtenons :

```
<Book>
        <Title Font=Verdana>The Best Seller</Title>
</Book>
```

Référence

DOM LIRE ATTRIBUT XML PAR INDEX, DOM LIRE ATTRIBUT XML PAR NOM.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

DOM ECRIRE NOM ELEMENT XML

DOM ECRIRE NOM ELEMENT XML (refElément; nomElément)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElément	Chaîne	→	Nouveau nom de l'élément

Description

La commande DOM ECRIRE NOM ELEMENT XML permet de modifier le nom de l'élément désigné par refElément.

Passez dans refElément la référence de l'élément à renommer et dans nomElément le nouveau nom de l'élément. Bien entendu, la commande se charge de modifier les balises d'ouverture et de fermeture de l'élément.

Exemple

Soit la source XML suivante :

```
<Book>
        <Title>The Best Seller</Title>
</Book>
```

Si le code suivant est exécuté, en admettant que vRefElem contienne la référence de l'élément 'Book' :

```
DOM ECRIRE NOM ELEMENT XML(vRefElem;"BestSeller")
```

Nous obtenons :

```
<BestSeller>
        <Title>The Best Seller</Title>
</BestSeller>
```

Référence

DOM LIRE NOM ELEMENT XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :
• la référence de l'élément n'est pas valide
• le nouveau nom d'élément n'est pas valide (par exemple, s'il débute par un chiffre).

DOM ECRIRE OPTIONS XML (refElément; encodage{; autonome{; indentation}})

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
encodage	Chaîne	→	Jeu de caractères du document XML
autonome	Booléen	→	Vrai=le document est autonome Faux (défaut)=le document n'est pas autonome
indentation	Booléen	→	Vrai (défaut)=le document est indenté Faux=le document n'est pas indenté

Description

La commande DOM ECRIRE OPTIONS XML permet de définir diverses options qui seront utilisée pour la création de l'arbre XML désigné par refElément. Ces options concernent l'encodage, l'attribut autonome (standalone) et l'indentation de l'arbre :

• encodage : indique le jeu de caractères employé. Par défaut (si la commande n'est pas appelée), le jeu de caractères UTF-8 (Unicode compressé) est utilisé.

• autonome : indique si l'arbre est autonome (Vrai) ou s'il dépend, pour son fonctionnement, de ressources externes (Faux). Par défaut (si la commande n'est pas appelée ou si le paramètre est omis), l'arbre n'est pas autonome.

• indentation : indique si l'arbre doit faire apparaître des indentations (Vrai) ou non (Faux) correspondant aux niveaux hiérarchiques des clés XML. Par défaut (si la commande n'est pas appelée ou si le paramètre est omis), l'arbre est indenté.

Exemple

Cet exemple définit l'encodage et l'option standalone de l'élément refElément :

```
DOM ECRIRE OPTIONS XML(refElément;"UTF-16";Vrai)
```

Référence

DOM Creer ref XML.

DOM ECRIRE VALEUR ELEMENT XML (refElément{; xChemin}; valeurElément{; *})

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
xChemin	Texte	→	Chemin XPath de l'élément XML
valeurElément	Chaîne | Variable	→	Nouvelle valeur de l'élément
*	*	→	Si passé : définir la valeur en CDATA

Description

La commande DOM ECRIRE VALEUR ELEMENT XML permet de modifier la valeur de l'élément désigné par refElément.

Si vous passez le paramètre facultatif xChemin, vous choisissez d'utiliser la notation XPath pour désigner l'élément à modifier (pour plus d'informations sur cette notation, reportez-vous au paragraphe "Utilisation de la notation XPath" dans la section Présentation des commandes XML DOM). Dans ce cas, vous devez passer la référence d'un élément XML racine dans refElément et le chemin XPath de l'élément à modifier dans xChemin.

Passez dans valeurElément une chaîne ou une variable (ou un champ) contenant la nouvelle valeur de l'élément :
• si vous passez une chaîne, la valeur sera affectée telle quelle dans la structure XML.
• si vous passez une variable ou un champ, 4D appliquera un traitement approprié à la valeur en fonction du type de valeurElément. Tous les types de données peuvent être utilisés, à l'exception des tableaux, images et pointeurs.

Lorsque le paramètre facultatif astérisque (*) est passé, vous indiquez que la valeur de l'élément doit être définie sous la forme CDATA. La forme spéciale CDATA permet d'écrire du texte sous forme brute (cf. exemple 2).

Note : Lorsque l'élément désigné par refElément est de type BLOB, DOM ECRIRE VALEUR ELEMENT XML l'encode automatiquement en base64. En revanche, la commande DOM LIRE VALEUR ELEMENT XML n'effectue pas automatiquement l'opération inverse, vous devez utiliser la commande DECODE.

Exemples

(1) Soit la source XML suivante :

```
<Book>
        <Title>The Best Seller</Title>
</Book>
```

Si le code suivant est exécuté, en admettant que vRefElem contienne la référence de l'élément 'Title' :

```
DOM ECRIRE VALEUR ELEMENT XML(vRefElem;"The Loser")
```

Nous obtenons :

```
<Book>
    <Title>The Loser</Title>
</Book>
```

(2) Soit la source XML suivante :

```
<Maths>
    <Postulate>1+2=3</Postulate>
</Maths>
```

Nous souhaitons écrire le texte "12 < 18" dans l'élément <postulate>. Cette chaîne ne peut pas être écrite telle quelle en XML car le caractère "<" n'est pas accepté. Ce caractère doit donc être transformé en "<", ou la forme CDATA doit être utilisée. Si vElemRef désigne le noeud XML <Postulate> :

```
    ` Forme normale
DOM ECRIRE VALEUR ELEMENT XML(vRefElem;"12 < 18")
```

Nous obtenons :

```
<Maths>
    <Postulate>12 &lt; 18</Postulate>
</Maths>
```

```
    ` Forme CDATA
DOM ECRIRE VALEUR ELEMENT XML(vRefElem;"12 < 18";*)
```

Nous obtenons :

```
<Maths>
    <Postulate><![CDATA[12 < 18]]></Postulate>
</Maths>
```

Référence

DOM LIRE VALEUR ELEMENT XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée (par exemple, si la référence de l'élément n'est pas valide).

DOM EXPORTER VERS FICHIER

DOM EXPORTER VERS FICHIER (refElément; cheminFichier)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML racine
cheminFichier	Texte	→	Chemin d'accès complet du fichier

Description

La commande DOM EXPORTER VERS FICHIER permet de sauvegarder un arbre XML dans un fichier sur disque.

Passez dans refElément la référence de l'élément racine à exporter.
Passez dans cheminFichier le chemin d'accès complet du fichier d'export à utiliser ou à créer.
Si le fichier n'existe pas, il est créé.
Si vous passez uniquement un nom de fichier (sans chemin d'accès), le fichier sera recherché ou créé à côté du fichier de structure.
Si vous passez une chaîne vide (""), une boîte de dialogue standard d'ouverture et de création de fichier apparaît.

Exemple

Cet exemple sauvegarde l'arbre vRefElem dans le fichier MonDoc.xml :

```
DOM EXPORTER VERS FICHIER(vRefElem;"C:\\dossier\MonDoc.xml")
```

Référence

DOM EXPORTER VERS IMAGE, DOM EXPORTER VERS VARIABLE.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

Gestion des erreurs

Une erreur est générée lorsque :
• la référence de l'élément n'est pas valide,
• le chemin d'accès spécifié n'est pas valide,
• le volume de stockage retourne une erreur (disque plein, etc.).

DOM EXPORTER VERS VARIABLE (refElément; vVarXml)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML racine
vVarXml	Texte | BLOB	→	Variable devant recevoir l'arbre XML

Description

La commande DOM EXPORTER VERS VARIABLE permet de sauvegarder un arbre XML dans une variable texte ou BLOB.

Passez dans refElément la référence de l'élément racine à exporter.

Passez dans vVarXml le nom de la variable devant contenir l'arbre XML. Cette variable peut être de type Texte ou BLOB. Vous pouvez choisir le type en fonction des opérations à effectuer par la suite ou de la taille que l'arbre peut atteindre (rappelons qu'en mode Non unicode les variables de type Texte sont limitées à 32 K de texte, en mode Unicode cette limite est de 2 Go).

Attention, en mode Non unicode si vous utilisez une variable Texte pour stocker l'élément refElément, il sera encodé en Mac "courant" (c'est-à-dire Mac Roman sur la plupart des systèmes occidentaux). Cela signifie que le texte retourné ne sera plus dans l'encodage d'origine (encoding="xxx"). Dans ce cas, la variable vVarXml permet de visualiser ou de stocker le code obtenu mais PAS de générer un document XML valide (via la commande ENVOYER PAQUET par exemple).

En mode Unicode, l'encodage d'origine est conservé dans la variable.

Exemple

Cet exemple sauvegarde l'arbre vRefElem dans une variable texte :

```
C_TEXTE(vtMonTexte)
DOM EXPORTER VERS VARIABLE(vRefElem;vtMonTexte)
```

Référence

DOM EXPORTER VERS FICHIER, DOM EXPORTER VERS IMAGE.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée (par exemple, si la référence de l'élément n'est pas valide).

DOM FERMER XML

version 2004 (Modifiée)

DOM FERMER XML (refElément)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML racine

Description

La commande DOM FERMER XML libère l'espace mémoire occupé par l'objet XML désigné par refElément.

Si refElément n'est pas un objet XML racine, une erreur est générée.

Référence

DOM Analyser source XML, DOM Analyser variable XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM LIRE ATTRIBUT XML PAR INDEX

version 2004 (Modifiée)

DOM LIRE ATTRIBUT XML PAR INDEX (refElément; indexAttribut; nomAttribut; valeurAttribut)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
indexAttribut	Entier long	→	Numéro d'indice de l'attribut
nomAttribut	Variable	←	Nom de l'attribut
valeurAttribut	Variable	←	Valeur de l'attribut

Description

La commande DOM LIRE ATTRIBUT XML PAR INDEX permet de connaître le nom ainsi que la valeur d'un attribut désigné par son numéro d'indice.

Passez dans refElément la référence d'un élément XML et dans indexAttribut le numéro d'indice de l'attribut dont vous voulez connaître le nom. Le nom est retourné dans le paramètre nomAttribut et sa valeur est retournée dans le paramètre valeurAttribut. 4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Note : Le numéro d'indice ne correspond pas à l'emplacement de l'attribut dans le fichier XML affiché sous forme de texte. En XML, l'indice d'un attribut indique sa position parmi les attributs classés par ordre alphabétique (en fonction de leur nom). Pour une illustration de ce principe, reportez-vous à l'exemple de la commande DOM Compter attributs XML.

Si la valeur passée dans indexAttribut est supérieure au nombre d'attributs présents dans l'élément XML, une erreur est retournée.

Exemple

Reportez-vous à l'exemple de la commande DOM Compter attributs XML.

Référence

DOM LIRE ATTRIBUT XML PAR NOM.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM LIRE ATTRIBUT XML PAR NOM

version 2004 (Modifiée)

DOM LIRE ATTRIBUT XML PAR NOM (refElément; nomAttribut; valeurAttribut)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomAttribut	Chaîne	→	Nom d'attribut
valeurAttribut	Variable	←	Valeur de l'attribut

Description

La commande DOM LIRE ATTRIBUT XML PAR NOM permet de connaître la valeur d'un attribut désigné par son nom.

Passez dans refElément la référence d'un élément XML et dans nomAttribut le nom d'attribut dont vous voulez connaître la valeur. La valeur est retournée dans le paramètre valeurAttribut.
4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Si aucun attribut nomAttribut n'existe dans l'élément XML, une erreur est retournée. Si plusieurs attributs de l'élément XML portent le nom spécifié, seule la valeur du premier attribut est retournée.

Exemple

Cette méthode permet de récupérer une valeur d'attribut XML à l'aide de son nom :

```
C_BLOB(maVarBlob)
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Enfant)
C_ENTIER LONG($NumLigne)

$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)
DOM LIRE ATTRIBUT XML PAR NOM($ref_XML_Enfant;"N";$NumLigne)
```

Si cette méthode est appliquée à l'exemple ci-dessous, *$NumLigne* contient la valeur 1 :

```
<?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

Référence

DOM LIRE ATTRIBUT XML PAR INDEX.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Lire dernier element XML enfant

DOM Lire dernier element XML enfant (refElément{; nomElémentEnf{; valeurElémentEnf}}) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElémentEnf	Chaîne	←	Nom de l'élément enfant
valeurElémentEnf	Chaîne	←	Valeur de l'élément enfant
Résultat	Chaîne	←	Référence de l'élément XML (16 caractères)

Description

La commande DOM Lire dernier element XML enfant retourne une référence XML vers le dernier "enfant" de l'élément XML passé en référence dans refElément. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres nomElémentEnf et valeurElémentEnf, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément enfant.

Exemple

Récupération de la référence du dernier élément XML du parent racine. La structure XML (C:\\import.xml) est préalablement chargée dans un BLOB :

```
C_BLOB(maVarBlob)
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Enfant)
C_TEXTE($nomEnfant;$valeurEnfant)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire dernier element XML enfant($ref_XML_Parent;
                                                $nomEnfant;$valeurEnfant)
```

Référence

DOM Lire premier element XML enfant.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Lire element XML

version 2004 (Modifiée)

DOM Lire element XML (refElément; nomElément; index; valeurElément) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElément	Chaîne	→	Nom de l'élément à lire
index	Entier long	→	Numéro d'indice de l'élément à lire
valeurElément	Variable	←	Valeur de l'élément
Résultat	Chaîne	←	Référence de l'élément XML (16 caractères)

Description

La commande DOM Lire element XML retourne une référence XML vers l'élément "enfant" dépendant des paramètres nomElément et index.

La valeur de l'élément est également retournée dans le paramètre valeurElément.

Référence

DOM LIRE VALEUR ELEMENT XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM Lire element XML frere precedent

version 2004.2

DOM Lire element XML frere precedent (refElément{; nomElémentFrère{; valeurElémentFrère}}) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElémentFrère	Chaîne	←	Nom de l'élément XML frère
valeurElémentFrère	Chaîne	←	Valeur de l'élément XML frère
Résultat	Chaîne	←	Référence de l'élément XML frère (16 caractères)

Description

La commande DOM Lire element XML frere precedent retourne une référence vers le précédent "frère" de l'élément XML passé en référence. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres nomElémentFrère et valeurElémentFrère, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément "frère" précédent.

Cette commande permet de naviguer parmi les "enfants" d'un élément XML.

Avant le premier "frère", la variable système OK prend la valeur 0.

Référence

DOM Lire element XML frere suivant.

Variables et ensembles système

Si la commande a été correctement exécutée et si l'élément référencé n'est pas le premier "enfant" de la structure, la variable système OK prend la valeur 1. Si une erreur se produit ou si l'élément analysé est le premier "enfant" de la structure, elle prend la valeur 0.

DOM Lire element XML frere suivant (refElément{; nomElémentFrère{; valeurElémentFrère}}) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElémentFrère	Chaîne	←	Nom de l'élément XML frère
valeurElémentFrère	Chaîne	←	Valeur de l'élément XML frère
Résultat	Chaîne	←	Référence de l'élément XML frère (16 caractères)

Description

La commande DOM Lire element XML frere suivant retourne une référence vers le prochain "frère" de l'élément XML passé en référence. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres nomElémentFrère et valeurElémentFrère, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément "frère".

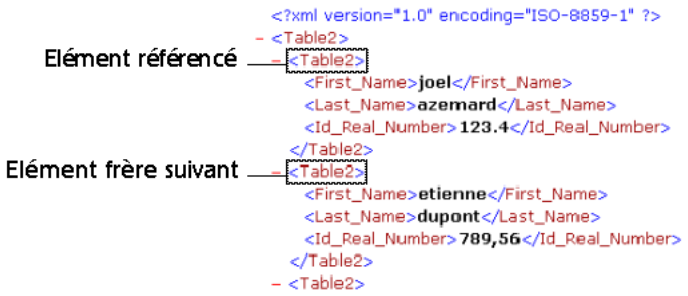
Cette commande permet de naviguer parmi les "enfants" d'un élément XML.

Après le dernier "frère", la variable système *OK* prend la valeur 0.

Exemples

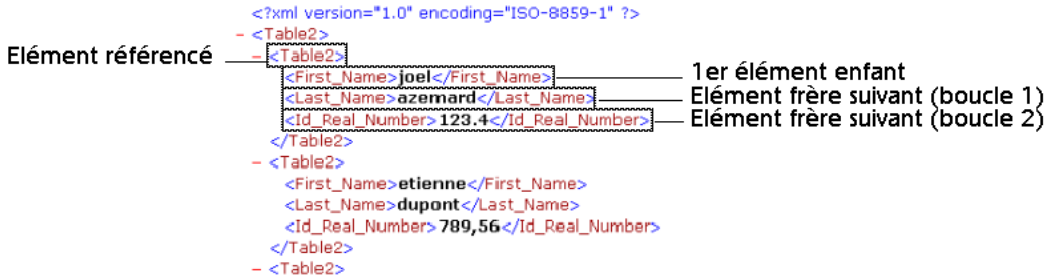
(1) Récupération de la référence de l'élément XML frère suivant l'élément passé en paramètre :

```
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Suivant)
$ref_XML_Suivant:=DOM Lire element XML frere suivant($ref_XML_Parent)
```



(2) Récupération dans une boucle des références de tous les éléments XML enfants de l'élément parent passé en paramètre, à compter du premier enfant :

```
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Premier;$ref_XML_Suivant)

$ref_XML_Premier:=DOM Lire premier element XML enfant($ref_XML_Parent)
$ref_XML_Suivant:=$ref_XML_Premier
Tant que (OK=1)
    $ref_XML_Suivant:=DOM Lire element XML frere suivant($ref_XML_Suivant)
Fin tant que
```



Référence

DOM Lire premier element XML enfant.

Variables et ensembles système

Si la commande a été correctement exécutée et si l'élément analysé n'est pas le dernier "frère" de l'élément référencé, la variable système *OK* prend la valeur 1. Si une erreur se produit ou si l'élément analysé est le dernier "frère" de l'élément référencé, elle prend la valeur 0.

DOM Lire element XML parent

DOM Lire element XML parent (refElément{; nomElémentPar{; valeurElémentPar}}) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElémentPar	Chaîne	←	Nom de l'élément XML parent
valeurElémentPar	Chaîne	←	Valeur de l'élément XML parent
Résultat	Chaîne	←	Référence de l'élément XML parent (16 caractères)

Description

La commande DOM Lire element XML parent retourne une référence XML vers le "parent" de l'élément XML passé en référence dans refElément. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres nomElémentPar et valeurElémentPar, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément parent.

Référence

DOM Lire dernier element XML enfant, DOM Lire element XML racine, DOM Lire premier element XML enfant.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Sinon, elle prend la valeur 0.

DOM Lire element XML racine

DOM Lire element XML racine (refElément) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
Résultat	Chaîne	←	Référence de l'élément racine (16 caractères) ou "" en cas d'erreur

Description

La commande DOM Lire element XML racine retourne une référence vers l'élément racine du document auquel appartient l'élément XML passé dans le paramètre refElément. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Référence

DOM Lire element XML parent.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, par exemple si la référence de l'élément est invalide, la variable OK prend la valeur 0 et la commande provoque une erreur. La commande retourne dans ce cas une chaîne vide.

DOM Lire informations XML (refElément; infoXML) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML racine
infoXML	Entier long	→	Type d'information à lire
Résultat	Chaîne	←	Valeur de l'information XML

Description

La commande DOM Lire informations XML permet de récupérer diverses informations sur l'élément XML désigné par refElément.

Passez dans infoXML un code indiquant le type d'information à récupérer. Vous pouvez utiliser les constantes prédéfinies suivantes, placées dans le thème "XML" :

Constante	Type	Valeur
ID PUBLIC	Entier long	1
ID SYSTEM	Entier long	2
Nom DOCTYPE	Entier long	3
Encoding	Entier long	4
Version	Entier long	5
URI Document	Entier long	6

Ces constantes indiquent les informations suivantes :
• *ID PUBLIC* : Identificateur public (FPI) de la DTD à laquelle le document se conforme (si la balise DOCTYPE xxx PUBLIC est présente).
• *ID SYSTEM* : Identificateur système.
• *Nom DOCTYPE* : Nom de l'élément racine tel que défini dans la balise DOCTYPE.
• *Encoding* : Encodage utilisé (UTF-8, ISO...).
• *Version* : Version de XML accepté.
• *URI Document* : URI de la DTD.

Référence

LIRE ERREUR XML.

Constantes

Thème XML.

DOM LIRE NOM ELEMENT XML

version 2004 (Modifiée)

DOM LIRE NOM ELEMENT XML (refElément; nomElément)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElément	Variable	←	Nom de l'élément

Description

La commande DOM LIRE NOM ELEMENT XML retourne dans le paramètre nomElément le nom de l'élément XML désigné par refElément. Pour plus d'informations sur les noms d'éléments XML, reportez-vous à la section Présentation des commandes XML DOM.

Exemple

Cette méthode retourne le nom de l'élément *$ref_XML_Elément* :

```
C_ALPHA(16;$ref_XML_Elément)
C_TEXTE($nom)

DOM LIRE NOM ELEMENT XML($ref_XML_Elément;$nom)
```

Référence

DOM ECRIRE NOM ELEMENT XML, DOM Lire element XML, DOM LIRE VALEUR ELEMENT XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

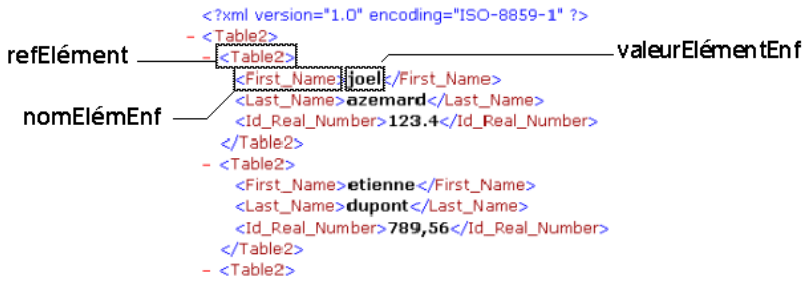
DOM Lire premier element XML enfant (refElément{; nomElémentEnf{; valeurElémentEnf}}) → Chaîne

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
nomElémentEnf	Chaîne	←	Nom de l'élément XML enfant
valeurElémentEnf	Chaîne	←	Valeur de l'élément XML enfant
Résultat	Chaîne	←	Référence de l'élément XML enfant (16 caractères)

Description

La commande DOM Lire premier element XML enfant retourne une référence XML vers le premier "enfant" de l'élément XML passé en référence dans refElément. Cette référence pourra être utilisée avec les autres commandes d'analyse XML.

Les paramètres nomElémentEnf et valeurElémentEnf, s'ils sont passés, reçoivent respectivement le nom et la valeur de l'élément enfant.



Exemples

(1) Récupération de la référence du premier élément XML du parent racine. La structure XML (C:\\import.xml) est préalablement chargée dans un BLOB :

```
C_BLOB(maVarBlob)
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Enfant)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent)
```

(2) Récupération de la référence, du nom et de la valeur du premier élément XML du parent racine. La structure XML (C:\\import.xml) est préalablement chargée dans un BLOB :

```
C_BLOB(maVarBlob)
C_ALPHA(16;$ref_XML_Parent;$ref_XML_Enfant)
C_TEXTE($enfantNom;$enfantValeur)

DOCUMENT VERS BLOB("c:\\import.xml";maVarBlob)
$ref_XML_Parent:=DOM Analyser variable XML(maVarBlob)
$ref_XML_Enfant:=DOM Lire premier element XML enfant($ref_XML_Parent;
                                                     $enfantNom;$enfantValeur)
```

Référence

DOM Lire element XML frere suivant.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Sinon, elle prend la valeur 0.

DOM LIRE VALEUR ELEMENT XML (refElément; valeurElément{; cDATA})

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
valeurElément	Variable	←	Valeur de l'élément
cDATA	Variable	←	Contenu de la section CDATA

Description

La commande DOM LIRE VALEUR ELEMENT XML retourne dans le paramètre valeurElément la valeur de l'élément XML désigné par refElément. 4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Le paramètre facultatif cDATA permet de récupérer le contenu de la ou des section(s) CDATA de l'élément XML refElément le cas échéant. Comme pour le paramètre valeurElément, 4D tentera de convertir la valeur obtenue dans le type de la variable passée en paramètre.

Note : Si l'élément désigné par refElément est un BLOB traité par la commande DOM ECRIRE VALEUR ELEMENT XML, il a été automatiquement encodé en base64. DOM LIRE VALEUR ELEMENT XML n'effectuant pas automatiquement l'opération inverse, vous devez utiliser la commande DECODE dans ce cas.

Exemple

Cette méthode retourne la valeur de l'élément *$ref_XML_Elément* :

```
C_ALPHA(16;$ref_XML_Elément)
C_REEL($valeur)

DOM LIRE VALEUR ELEMENT XML($ref_XML_Elément;$valeur)
```

Référence

DOM ECRIRE VALEUR ELEMENT XML, DOM Lire element XML, DOM LIRE NOM ELEMENT XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

DOM SUPPRIMER ELEMENT XML

DOM SUPPRIMER ELEMENT XML (refElément)

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML

Description

La commande DOM SUPPRIMER ELEMENT XML supprime l'élément désigné par refElément.

Référence

DOM Creer element XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.
Une erreur est générée lorsque :
• la référence de l'élément n'est pas valide,
• l'élément est vide.

66

XML SAX

Présentation des commandes XML SAX

Ce thème regroupe les commandes XML SAX de 4D.

Pour des informations générales sur le XML (présentation, jeux de caractères, glossaire) ainsi que sur les différences entre les modes DOM et SAX, veuillez vous reporter à la section Présentation des commandes XML DOM.

Création, ouverture et fermeture des documents XML via SAX
Les commandes SAX travaillent avec des références de documents standard de 4D (DocRef, référence de type Heure). Il est donc possible d'utiliser ces commandes conjointement avec les commandes 4D permettant de gérer les documents, par exemple ENVOYER PAQUET ou Ajouter a document.
La création et l'ouverture par programmation de documents XML est effectuée via les commandes Creer document et Ouvrir document. Par la suite, l'utilisation d'une commande XML avec ces documents provoquera la mise en oeuvre automatique des mécanismes XML tels que l'encodage. Par exemple, l'en-tête <?xml version="1.0" encoding="… encodage …" standalone = "no "?> sera automatiquement écrit dans le document.

Note : Les documents destinés à la lecture SAX doivent impérativement être ouverts en mode "lecture seule" par la commande Ouvrir document. Ce principe est destiné à prévenir les conflits pouvant survenir entre 4D et la librairie Xerces lors de l'ouverture simultanée de documents "standard" et de documents XML. Si vous exécutez une commande d'analyse SAX avec un document ouvert en lecture-écriture, un message d'alerte est affiché et l'analyse est impossible.

La fermeture d'un document XML SAX doit être effectuée à l'aide de la commande FERMER DOCUMENT. Si des éléments XML étaient ouverts, ils sont automatiquement refermés.

SAX AJOUTER CDATA XML (document; données)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
données	Texte | BLOB	→	Texte ou BLOB à insérer dans le document entre balises CData

Description

La commande SAX AJOUTER CDATA XML ajoute dans le document XML référencé par document des données de type texte ou BLOB. Ces données seront automatiquement encadrées par les balises <![CDATA[et]]>.
Le texte compris dans une section CData est ignoré par l'interpréteur XML.

Si vous souhaitez encoder le contenu de données, vous devez utiliser la commande ENCODER. Dans ce cas bien entendu, vous devez passer un BLOB dans données.

Pour que cette commande fonctionne, un élément doit être ouvert. Dans le cas contraire, une erreur est générée.

Exemple

Vous souhaitez insérer les lignes suivantes dans votre document XML :

```
function matchwo(a,b)
{
if (a < b && a < 0) then
                {
                return 1
                }
else
                {
                return 0
                }
}
```

Pour cela, il vous suffit d'exécuter le code suivant :

```
C_TEXTE (vtMontexte)
... ` placez ici le texte dans la variable vtMontexte
SAX AJOUTER CDATA XML($RefDoc;vtMontexte)
```

Le résultat sera alors :

```
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
        {
        return 1
        }
else
        {
        return 0
        }
}
]]>
```

Référence

SAX LIRE CDATA XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

SAX AJOUTER COMMENTAIRE XML (document; commentaire)

Paramètre	Type		Description
document	DocRef	-->	Référence du document ouvert
commentaire	Chaîne	-->	Commentaire à ajouter

Description

La commande SAX AJOUTER COMMENTAIRE XML ajoute un commentaire dans le document XML référencé par document.

Un commentaire XML est un texte dont le contenu ne sera pas analysé par l'interpréteur XML. Les commentaires XML sont encadrés par les caractères <!-- et -->.

Exemple

L'instruction suivante :

```
vCommentaire := "Créé par 4D"
SAX AJOUTER COMMENTAIRE XML ($RefDoc;vCommentaire)
```

... inscrira cette ligne dans le document :

```
<!--Créé par 4D-->
```

Référence

SAX AJOUTER DOCTYPE XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

En cas d'erreur, la commande retourne une erreur qui peut être interceptée via une méthode d'appel sur erreur.

SAX AJOUTER DOCTYPE XML (document; docType)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
docType	Chaîne	→	DocType à ajouter

Description

La commande SAX AJOUTER DOCTYPE XML ajoute l'instruction DocType définie par le paramètre docType dans le document XML référencé par document.

Une instruction DocType permet d'indiquer le type de XML dans lequel le document a été écrit et de désigner la Déclaration de type de document (DTD) utilisée. Une instruction DocType est généralement de la forme <!DOCTYPE type_XML "adresse_DTD">.

Exemple

L'instruction suivante :

```
vDocType := "Livres SYSTEM \"Livre.DTD\""
SAX AJOUTER DOCTYPE XML ($RefDoc;vDocType)
```

... inscrira cette ligne dans le document :
<!DOCTYPE Livres SYSTEM "Livre.DTD">

Référence

SAX AJOUTER COMMENTAIRE XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

Gestion des erreurs

En cas d'erreur, la commande retourne une erreur qui peut être interceptée via une méthode d'appel sur erreur.

SAX AJOUTER INSTRUCTION DE TRAITEMENT (document; instruction)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
instruction	Texte	→	Instruction à insérer dans le document

Description

La commande SAX AJOUTER INSTRUCTION DE TRAITEMENT ajoute dans le document XML référencé par document une instruction de traitement XML.

Une instruction de traitement permet d'indiquer le type d'application et éventuellement des paramètres additionnels permettant de traiter une entité externe non analysable.

La commande formate les données d'instruction conformément au XML. En revanche, les instructions elles-mêmes ne sont pas analysées, il revient au développeur de s'assurer qu'elles sont valides.

Exemple

Le code suivant :

```
vtInstruct:="xml-stylesheet type="+Caractere(Guillemets)+"text/xsl"+
                                    Caractere (Guillemets)+"href="+Caractere (Guillemets)+
                                    "style.xsl"+Caractere (Guillemets)
SAX AJOUTER INSTRUCTION DE TRAITEMENT ($RefDoc;vtInstruct)
```

... inscrira cette ligne dans le document :

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Référence

SAX LIRE INSTRUCTION DE TRAITEMENT XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX AJOUTER VALEUR ELEMENT XML

version 11 (Modifiée)

SAX AJOUTER VALEUR ELEMENT XML (document; données{; *})

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
données	Texte | Variable	→	Texte ou variable à insérer dans le document
*	*	→	Si passé = Encoder les caractères spéciaux Si omis = Ne pas encoder les caractères spéciaux

Description

La commande SAX AJOUTER VALEUR ELEMENT XML ajoute directement dans le document XML référencé par document des données sans les convertir. Cette commande équivaut par exemple à insérer une pièce jointe dans le corps (body) d'un email.

Vous pouvez passer dans le paramètre données soit directement une chaîne de caractères, soit une variable 4D. Le contenu de la variable sera converti en texte pour pouvoir être inséré dans le document XML.
Si vous souhaitez encoder le contenu de données, vous devez utiliser la commande ENCODER. Dans ce cas bien entendu, vous devez passer un BLOB dans données.

Par défaut, la commande n'encode pas les caractères spéciaux (< > " '...) contenus dans le paramètre données. Pour forcer l'encodage de ces paramètres, il suffit de passer le paramètre facultatif *.

Pour que cette commande fonctionne, un élément doit être ouvert. Dans le cas contraire, une erreur est générée.

Exemple

Cet exemple insère le fichier whitepaper.pdf dans l'élément XML ouvert :

```
C_BLOB(vBMonBLOB)
DOCUMENT VERS BLOB ("c:\\whitepaper.pdf";vBMonBLOB)
SAX AJOUTER VALEUR ELEMENT XML($RefDoc;vBMonBLOB)
```

Référence

SAX LIRE VALEUR ELEMENT XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX ECRIRE OPTIONS XML (document; encodage{; autonome{; indentation}})

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
encodage	Chaîne	→	Jeu de caractères du document XML
autonome	Booléen	→	Vrai=le document est autonome Faux (défaut)=le document n'est pas autonome
indentation	Booléen	→	Vrai (défaut)=le document est indenté Faux=le document n'est pas indenté

Description

La commande SAX ECRIRE OPTIONS XML initialise le document XML référencé par document à l'aide des valeurs passées en paramètres. Ces paramètres permettent de déterminer l'encodage, l'attribut autonome (standalone) et l'indentation du document.
• encodage : indique le jeu de caractères employé dans le document. Par défaut (si la commande n'est pas appelée), le jeu de caractères UTF-8 (Unicode compressé) est utilisé.
• autonome : indique si le document est autonome (Vrai) ou s'il dépend, pour son fonctionnement, d'autres fichiers ou de ressources externes (Faux). Par défaut (si la commande n'est pas appelée ou si le paramètre est omis), le document n'est pas autonome.
• indentation : indique si le document doit faire apparaître des indentations (Vrai) ou non (Faux) correspondant aux niveaux hiérarchiques des clés XML. Par défaut (si la commande n'est pas appelée ou si le paramètre est omis), le document est indenté.

Cette commande doit être appelée une seule fois par document et avant la première commande d'écriture XML dans le document, sinon une erreur est générée.

Exemple

Le code suivant :

```
SAX ECRIRE OPTIONS XML($RefDoc;"UTF-16";Vrai)
```

... inscrira cette ligne dans le document :

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
```

Référence

SAX LIRE VALEURS DOCUMENT XML.

SAX FERMER ELEMENT XML (document)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert

Description

La commande SAX FERMER ELEMENT XML inscrit dans le document XML référencé par document les instructions nécessaires à la fermeture du dernier élément ouvert via la commande SAX OUVRIR ELEMENT XML.

L'emploi de cette commande est facultatif. En effet, 4D ajoute automatiquement si nécessaire, au moment de la fermeture des documents XML, les balises de fin d'éléments non refermés explicitement.

Exemple

Si le dernier élément ouvert est <Book>, l'instruction suivante :

```
SAX FERMER ELEMENT XML($RefDoc)
```

... inscrira cette ligne dans le document :

</Book>

Référence

SAX OUVRIR ELEMENT XML, SAX OUVRIR ELEMENT XML TABLEAUX.

SAX LIRE CDATA XML (document; valeur)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
valeur	Texte | BLOB	←	Valeur de l'élément

Description

La commande SAX LIRE CDATA XML permet de récupérer la valeur CDATA d'un élément XML existant dans le document XML référencé par document. Elle doit être appelée dans le contexte d'un événement SAX CDATA XML. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

Passez une variable valeur de type Texte si vous souhaitez récupérer des données de taille supérieure à 32 Ko (la base doit fonctionner en mode Unicode).

Les données sont retournées telles quelles (elles ne sont pas modifiées).

Exemple

Considérons l'extrait de code XML suivant :

```
<RootElement>
        <Child>MonTexte<![CDATA[MonCData]]</Child>
</RootElement>
```

Le code 4D suivant retournera "MonCData" dans vDonnéesTexte :

```
C_BLOB (vDonnées)
C_TEXTE (vDonnéesTexte)
SAX LIRE CDATA XML(RefDoc;vDonnées)
vDonnéesTexte:=BLOB vers texte(vDonnées;UTF8 Chaîne en C)
```

Référence

SAX AJOUTER CDATA XML, SAX Lire noeud XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE COMMENTAIRE XML

SAX LIRE COMMENTAIRE XML (document; commentaire)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
commentaire	Chaîne	←	Commentaire XML

Description

La commande SAX LIRE COMMENTAIRE XML retourne un commentaire si un événement SAX de type Commentaire XML est généré dans le document XML référencé par document. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

Référence

SAX AJOUTER COMMENTAIRE XML, SAX Lire noeud XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE ELEMENT XML (document; nom; préfixe; nomsAttributs; valeursAttributs)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
nom	Chaîne	←	Nom de l'élément
préfixe	Chaîne	←	Espace de nommage
nomsAttributs	Tab chaîne	←	Noms des attributs
valeursAttributs	Tab chaîne	←	Valeurs des attributs

Description

La commande SAX LIRE ELEMENT XML retourne diverses informations relatives à l'élément nom présent dans le document XML référencé par document. Elle doit être appelée dans le contexte d'un événement SAX Début élément XML ou Fin élément XML. Dans le cas particulier d'un Fin élément XML, les paramètres d'attributs ne sont pas gérés. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

nom contient le nom de l'élément.

préfixe retourne l'espace de nommage (*namespace*) de l'élément. Ce paramètre est vide si aucun espace de nommage n'est associé à l'élément.

La commande remplit le tableau nomsAttributs avec les noms des attributs de l'élément cible. Si nécessaire, la commande crée et dimensionne automatiquement le tableau.

La commande remplit également le tableau valeursAttributs avec les valeurs des attributs de l'élément cible. Si nécessaire, la commande crée et dimensionne automatiquement le tableau.

Exemple

Considérons l'extrait de code XML suivant :

```
<RootElement>
        <Child Att1="111" Att2="222" Att3="333">MonTexte</Child>
</RootElement>
```

Une fois l'instruction suivante exécutée :

```
SAX LIRE ELEMENT XML (RefDoc;vNom;vPréfixe;tAttrNoms;tAttrValeurs)
```

...vNom contiendra “Child”
vPréfixe contiendra “”
tAttrNoms{1} contiendra “Att1”, tAttrNoms{2} contiendra “Att2”, tAttrNoms{3} contiendra “Att3”
tAttrValeurs{1} contiendra “111”, tAttrValeurs{2} contiendra “222”, tAttrValeurs{3} contiendra “333”

Référence

SAX Lire noeud XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE ENTITE XML (document; nom; valeur)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
nom	Chaîne	←	Nom de l'entité
valeur	Chaîne	←	Valeur de l'entité

Description

La commande SAX LIRE ENTITE XML permet de récupérer le nom et valeur d'une entité XML présente dans le document XML référencé par document. Elle doit être appelée dans le contexte d'un événement SAX Entité XML. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

Exemples

Considérons l'extrait de code XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE body [
        <!ELEMENT body (element*)>
        <!ELEMENT element (#PCDATA)>
        <!ENTITY nom "Le remplacement">
]>
<body>
        <element>L'entité est mise à jour par &nom;</element>
</body>
```

L'instruction suivante retournera "nom" dans vNom et "Le remplacement" dans vValeur.

```
SAX LIRE ENTITE XML(RefDoc;vNom;vValeur)
```

Référence

SAX Lire noeud XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE INSTRUCTION DE TRAITEMENT XML

SAX LIRE INSTRUCTION DE TRAITEMENT XML (document; nom; valeur)

Paramètre	Type		Description
document	DocRef	-->	Référence du document ouvert
nom	Chaîne	←	Nom de l'instruction
valeur	Chaîne	←	Valeur de l'instruction

Description

La commande SAX LIRE INSTRUCTION DE TRAITEMENT XML retourne le nom et la valeur de l'instruction de traitement XML analysée dans le document XML référencé par document. Cette commande doit être appelée dans le contexte d'un événement Instruction de traitement XML. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

Exemple

Considérons l'extrait de code XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Myself (4D SA)-->
<?PI TextProcess?>
<!DOCTYPE RootElement SYSTEM "ParseTest.dtd">
```

L'instruction suivante retournera "PI" dans vNom et "TextProcess" dans vValeur :

```
SAX LIRE INSTRUCTION DE TRAITEMENT XML($RefDoc;vNom;vValeur)
```

Référence

SAX AJOUTER INSTRUCTION DE TRAITEMENT, SAX Lire noeud XML.

SAX Lire noeud XML (document) → Entier long

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
Résultat	Entier long	←	Evénement retourné par la fonction

Description

La commande SAX Lire noeud XML retourne un entier long indiquant le type d'événement SAX retourné durant l'analyse du document XML référencé par document.

Les événements pouvant être retournés sont fournis sous forme de constantes dans le thème "XML" :

Constante	Type	Valeur
Début document XML	Entier long	1
Commentaire XML	Entier long	2
Instruction de traitement XML	Entier long	3
Début élément XML	Entier long	4
Fin élément XML	Entier long	5
Donnée XML	Entier long	6
CDATA XML	Entier long	7
Entité XML	Entier long	8
Fin document XML	Entier long	9

Exemple

Exemple de traitement des événements :

```
    `Ouverture en lecture seule obligatoire
RefDoc:=Ouvrir document("";"xml";Mode lecture)
Si (OK=1)
   Repeter
      MonEvénement:=SAX Lire noeud XML(RefDoc)
      Au cas ou
         : (MonEvénement=Début document XML)
            FaireQuelqueChose
```

```
            : (MonEvénement=Commentaire XML)
                FaireAutreChose
        Fin de cas
    Jusque (MonEvénement=Fin document XML)
Fin de si
FERMER DOCUMENT (RefDoc)
```

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE VALEUR ELEMENT XML (document; valeur)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
valeur	Texte | BLOB	←	Valeur de l'élément

Description

La commande SAX LIRE VALEUR ELEMENT XML permet de récupérer la valeur d'un élément XML existant dans le document XML référencé par document. Elle doit être appelée dans le contexte d'un événement SAX Donnée XML. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

Passez dans le paramètre valeur une variable de type Texte ou BLOB devant récupérer les données. Si vous passez un BLOB, le texte sera retourné tel quel (il ne sera pas modifié).

Exemple

Considérons l'extrait de code XML suivant :

```
<RootElement>
        <Child Att1="111" Att2="222" Att3="333">MonTexte</Child>
</RootElement>
```

L'instruction suivante retournera "MonTexte" dans vValeur :

```
SAX LIRE VALEUR ELEMENT XML(RefDoc;vValeur)
```

Référence

SAX AJOUTER VALEUR ELEMENT XML, SAX Lire noeud XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX LIRE VALEURS DOCUMENT XML

SAX LIRE VALEURS DOCUMENT XML (document; encodage; version; autonome)

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
encodage	Chaîne	←	Jeu de caractères du document XML
version	Chaîne	←	Version du XML
autonome	Booléen	←	Vrai=le document est autonome, sinon Faux

Description

La commande SAX LIRE VALEURS DOCUMENT XML extrait des informations élémentaires de l'en-tête XML du document XML référencé par document.

La commande retourne respectivement le type d'encodage, la version et la propriété "autonome" du document dans les paramètres encodage, version et autonome. Cette commande doit être utilisée dans le contexte de l'événement SAX Début document XML. Pour plus d'informations sur les événements SAX, reportez-vous à la description de la commande SAX Lire noeud XML.

Référence

SAX ECRIRE OPTIONS XML, SAX Lire noeud XML.

Variables et ensembles système

Si la commande a été exécutée correctement, la variable système OK prend la valeur 1, sinon elle prend la valeur 0 et une erreur est générée.

SAX OUVRIR ELEMENT XML (document; balise{; nomAttribut; valeurAttribut}{; nomAttribut2; valeurAttribut2; ...; nomAttributN; valeurAttributN})

Paramètre	Type		Description
document	DocRef	→	Référence du document ouvert
balise	Chaîne	→	Nom de l'élément à ouvrir
nomAttribut	Chaîne	→	Nom d'attribut
valeurAttribut	Chaîne	→	Valeur d'attribut

Description

La commande SAX OUVRIR ELEMENT XML permet d'ajouter un nouvel élément dans le document XML référencé par document ainsi que, facultativement, des attributs et leurs valeurs.

L'élément ajouté est "ouvert" dans le document (la balise de fin n'est pas ajoutée). Pour refermer un élément créé à l'aide de cette commande, vous devez soit :
• utiliser la commande SAX FERMER ELEMENT XML,
• refermer le document XML. Dans ce cas, 4D ajoute automatiquement les balises XML de fermeture nécessaires.

Passez dans balise le nom de l'élément à créer. Ce nom peut contenir uniquement des lettres, des chiffres, ainsi que les caractères ".", "-","_" et ":". Si un caractère invalide est passé dans balise, une erreur est générée.

Facultativement, la commande permet de passer un ou plusieurs couple(s) attributs/valeurs (sous forme de variables, champs ou valeur littérales) via les paramètres nomAttribut et valeurAttribut. Vous pouvez passer autant de couples attribut/valeur que vous voulez.

Exemple

L'instruction suivante :

```
vElement:="Book"
SAX OUVRIR ELEMENT XML($RefDoc;vElement)
```

... inscrira cette ligne dans le document :

<Book

Référence

SAX FERMER ELEMENT XML, SAX OUVRIR ELEMENT XML TABLEAUX.

Gestion des erreurs

Si un caractère invalide est passé dans balise, une erreur est générée.

SAX OUVRIR ELEMENT XML TABLEAUX (document; balise{; tabNomsAttributs; tabValeursAttributs}{; tabNomsAttributs2; tabValeursAttributs2; ...; tabNomsAttributsN; tabValeursAttributsN})

Paramètre	Type		Description
document	docRef	→	Référence du document ouvert
balise	Chaîne	→	Nom de l'élément à ouvrir
tabNomsAttributs	Tab Chaîne	→	Tableau de noms d'attributs
tabValeursAttributs	Tab Chaîne	→	Tableau de valeurs d'attributs

Description

La commande SAX OUVRIR ELEMENT XML TABLEAUX permet d'ajouter un nouvel élément dans le document XML référencé par document ainsi que, facultativement, des attributs et leurs valeurs sous forme de tableaux.

Hormis la prise en charge de tableaux (cf. ci-dessous), cette commande est identique à SAX OUVRIR ELEMENT XML. Reportez-vous à la description de cette commande pour le détail de son fonctionnement.

Facultativement, la commande SAX OUVRIR ELEMENT XML TABLEAUX permet de passer plusieurs couples d'attributs et de valeurs d'attributs sous forme de tableaux dans les paramètres tabNomsAttributs et tabValeursAttributs.
Les tableaux doivent avoir été créés au préalable et fonctionner par paires. Vous pouvez passer autant de couples de tableaux et autant d'éléments dans chaque couple que vous voulez.

Exemple

La méthode suivante :

```
TABLEAU ALPHA(80;tNomsAtt;2)
TABLEAU ALPHA(80;tValeursAtt;2)
vElement:="Book"
tNomsAtt{1}:="Font"
tValeursAtt{1}:="arial"
tNomsAtt{2}:="Style"
tValeursAtt{2}:="Bold"
SAX OUVRIR ELEMENT XML TABLEAUX($RefDoc;vElement;tNomsAtt;tValeursAtt)
```

... inscrira cette ligne dans le document :

```
<Book Font="arial" Style="Bold">
```

Référence

SAX FERMER ELEMENT XML, SAX OUVRIR ELEMENT XML.

67

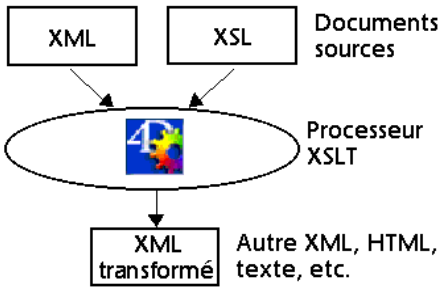
XML Utilitaires

Ce thème regroupe les commandes XML "utilitaires" de 4D. Il s'agit des commandes de gestion d'erreurs ainsi que des commandes spécialisées dans le XSL et le SVG.

Pour des informations générales sur le XML (présentation, glossaire) ainsi que sur les différences entre les modes DOM et SAX, veuillez vous reporter à la section Présentation des commandes XML DOM.

Prise en charge des transformations XSL
4D prend en charge l'application de feuilles de style XSL (*eXtended Stylesheet Language*). Le langage XSL permet de modifier le balisage d'un document XML.

Le langage XSL comporte deux facettes :
• **le formatage** : il permet d'appliquer des règles de style et d'affichage pour les éléments XML, un peu à l'image des CSS (*Cascading StyleSheet*) du langage HTML.
• **la transformation** : il permet de transformer un balisage XML en un autre système de balisage, par exemple en HTML. Cette fonction de transformation est plus spécifiquement appelée **XSLT**. Une feuille de style XSL peut entièrement réorganiser les éléments XML d'un document en les sélectionnant puis en les transformant en d'autres éléments. Cette fonction est utile par exemple pour harmoniser un ensemble de documents XML disparates.



Note : 4D utilise la librairie Xalan-C_1_6_0.dll pour la réalisation des transformations XSL. Xalan est un processeur XSLT du domaine public. Pour plus d'informations, reportez-vous à l'adresse http://xml.apache.org/xalan-c/index.html.

Les feuilles de style XSL sont des documents texte (extension ".xsl") générés manuellement ou à l'aide d'applications spécialisées. Le langage XSL comporte divers éléments et fonctions permettant d'effectuer tout type de transformation dynamique. Pour plus d'informations sur ce langage, reportez-vous au site http://xmlfr.org (par exemple).

4D vous permet de transformer un document XML à l'aide d'une feuille de style XSL existante (commande APPLIQUER TRANSFORMATION XSLT). En outre, 4D vous permet de modifier à la volée des paramètres de la feuille de style XSL via la commande FIXER PARAMETRE XSLT.

Note : Une option de la boîte de dialogue d'exportation permet également d'utiliser une feuille de style XSL lors d'un export XML et donc de générer un document XML transformé.

Qu'est-ce que le SVG ?
SVG (*Scalable Vector Graphics*) est un format de fichier permettant de décrire en XML un graphique vectoriel (extension .svg). L'utilisation la plus courante du SVG est la publication de données statistiques ou cartographiques.
Ces fichiers peuvent être visualisés dans les navigateurs Web, soit nativement, soit via des plug-ins. 4D v11 comporte un moteur de rendu SVG vous permettant de visualiser les fichiers SVG dans les champs ou les variables image. La commande SVG EXPORTER VERS IMAGE vous permet de générer une image dans 4D à partir d'une description SVG. A noter également que la commande GRAPHE permet de tirer parti du moteur SVG intégré de 4D.
Pour plus d'informations sur ce format, reportez-vous à l'adresse http://www.w3.org/Graphics/SVG/.

APPLIQUER TRANSFORMATION XSLT (sourceXML; feuilleXSL; résultat{; compileFeuille})

Paramètre	Type		Description
sourceXML	Chaîne | BLOB	→	Nom ou chemin d'accès du document XML source ou BLOB contenant le XML source
feuilleXSL	Chaîne | BLOB	→	Nom ou chemin d'accès du document contenant la feuille de style XSL, ou BLOB contenant la feuille de style XSL
résultat	Chaîne | BLOB	→	Nom ou chemin d'accès du document recevant le résultat de la transformation XSLT, ou BLOB recevant le résultat de la transformation XSLT
compileFeuille	Booléen	→	Vrai : optimise la transformation XSLT Faux ou omis : pas d'optimisation, efface le fichier XSL compilé s'il existe

Description

La commande APPLIQUER TRANSFORMATION XSLT applique une transformation XSL à un document ou un BLOB contenant du XML et génère un document ou un BLOB *résultat*. La portée de cette commande est le process courant.

Note : Pour plus d'informations sur la transformation XSL (ou *XSLT*), reportez-vous à la section Présentation des commandes XML Utilitaires.

La commande requiert trois paramètres de type BLOB ou Chaîne de caractères. **Attention**, la commande accepte uniquement des variables ou des champs comme paramètres.

Si vous passez une chaîne de caractères, vous désignez un document. Dans ce cas, vous pouvez passer uniquement le nom (le document doit se trouver à côté de la structure de la base de données) ou le chemin d'accès complet du document.
Il n'est pas possible de mixer différents types de paramètres au sein d'un même appel.

• Le paramètre sourceXML doit contenir le source XML à transformer. La commande vérifie la validité du code XML.
• Le paramètre feuilleXSL doit contenir la feuille de style XSL à utiliser pour la transformation XSLT. Cette feuille de style peut avoir été générée manuellement ou via un logiciel spécialisé. La commande vérifie la validité du code XML.
• Le paramètre résultat doit contenir le nom du document ou du BLOB devant recueillir le résultat de la transformation XSLT. Si vous passez un nom de document n'existant pas à l'emplacement désigné, 4D le crée automatiquement. Si le document est déjà ouvert en écriture, une erreur est générée.

La commande analyse le source XML et le transforme à l'aide des instructions de la feuille de style XSL. Si la commande FIXER PARAMETRE XSLT a été utilisée au préalable, la commande remplace les paramètres définis par leur valeur. Le résultat de la transformation est écrit dans le document ou BLOB résultat.

Le paramètre facultatif compileFeuille permet d'optimiser la transformation XSLT, notamment en cas d'applications successives de la même feuille XSL. Lorsque le paramètre compileFeuille est passé et vaut Vrai, le fichier XSL feuilleXSL est analysé au premier appel de la commande puis est compilé et stocké en mémoire. A chaque appel suivant avec le même fichier XSL, la commande utilise directement le fichier compilé (sauf s'il a été modifié), ce qui permet d'accélérer les traitements. L'optimisation ne prend pas en compte les éventuelles modifications effectuées dans les fichiers importés (via xsl:import). Si un fichier référencé par le fichier XSL est modifié, il est nécessaire de "forcer" la recompilation du nouveau fichier XSL en rappelant la commande avec le paramètre compileFeuille à Faux (ou omis).

Exemple

Reportez-vous à l'exemple de la commande FIXER PARAMETRE XSLT.

Référence

FIXER PARAMETRE XSLT, LIRE ERREUR XSLT.

Variables et ensembles système

Si la transformation a été correctement effectuée, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

FIXER PARAMETRE XSLT (nomParam; valeurParam)

Paramètre	Type		Description
nomParam	Chaîne	→	Nom du paramètre à chercher dans la feuille XSL
valeurParam	Chaîne	→	Valeur du paramètre à utiliser dans le document transformé

Description

La commande FIXER PARAMETRE XSLT doit être utilisée conjointement avec la commande APPLIQUER TRANSFORMATION XSLT. Elle permet de définir les valeurs de paramètres variables placés dans une feuille de style XSL au moment de la transformation XSLT d'un document XML. A l'aide de cette commande, il est notamment possible d'insérer des valeurs issues de traitements 4D dans la feuille de style XSL, juste avant son utilisation par APPLIQUER TRANSFORMATION XSLT.

Notes :
• La portée de cette commande est le process courant. Elle doit être appelée dans le même process que la commande APPLIQUER TRANSFORMATION XSLT associée.
• Pour plus d'informations sur la transformation XSL (ou *XSLT*), reportez-vous à la section Présentation des commandes XML Utilitaires.

Passez dans nomParam le nom du paramètre XSL variable à remplacer. Ce paramètre doit être présent dans la feuille de style XSL sous la forme $aremplacer. En revanche, le caractère $ n'est pas nécessaire dans nomParam. Par exemple, si l'instruction <xsl:template match=$mavar> est placée dans le fichier XSL, il vous suffira de passer "mavar" dans nomParam pour désigner ce paramètre.

Passez dans valeurParam la valeur que vous souhaitez insérer dans le fichier transformé en lieu et place du paramètre XSL variable. Pour reprendre l'exemple précédent, si vous passez la valeur "titre" dans valeurParam, la transformation XSLT prendra en compte l'instruction <xsl:template match="titre"> (ce qui désigne les éléments "titre" comme sujets de l'application d'une règle de style).
Si la valeur est de type chaîne, vous devez l'encadrer par des apostrophes (par exemple 'mavaleur') — en plus des guillemets de la syntaxe 4D ("'mavaleur'").

Note : Pour une description détaillée du langage XSL, vous pouvez vous reporter aux nombreux sites Web qui lui sont consacrés, par exemple http://xmlfr.org.

Pour passer plusieurs paramètres à une feuille de style XSL, il suffit d'appeler plusieurs fois la commande FIXER PARAMETRE XSLT. Les paramètres sont "empilés" jusqu'à l'appel de APPLIQUER TRANSFORMATION XSLT dans le même process. A l'issue de l'exécution de APPLIQUER TRANSFORMATION XSLT, la "pile" de paramètres est automatiquement effacée.

Exemple

L'exemple suivant définit deux paramètres XSL puis transforme le document mondoc.xml en fichier html via la feuille de style mafeuille.xsl :

```
FIXER PARAMETRE XSLT("varstyle";"'gras'")
FIXER PARAMETRE XSLT("varcouleur";"'bleu'")
$docxml:="mondoc.xml"
$feuillexsl:="mafeuille.xsl"
$dochtml:="mondoc.html"
APPLIQUER TRANSFORMATION XSLT($docxml;$feuillexsl;$dochtml)
```

Référence

APPLIQUER TRANSFORMATION XSLT, LIRE ERREUR XSLT.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1, sinon elle prend la valeur 0.

LIRE ERREUR XML (refElément; texteErreur{; ligne{; colonne}})

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML
texteErreur	Variable	←	Texte de l'erreur
ligne	Variable	←	Numéro de ligne
colonne	Variable	←	Numéro de colonne

Description

La commande LIRE ERREUR XML retourne dans le paramètre texteErreur la description de l'erreur rencontrée lors du traitement de l'élément XML désigné par le paramètre refElément. Les informations retournées sont fournies par la librairie Xerces.dll.

Les paramètres optionnels ligne et colonne désignent précisément l'emplacement de l'erreur : ils récupèrent respectivement le numéro de la ligne et, dans cette ligne, la position du premier caractère de l'expression à l'origine de l'erreur.

Référence

DOM Lire informations XML.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système *OK* prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

LIRE ERREUR XSLT (texteErreur{; ligne{; colonne}})

Paramètre	Type		Description
texteErreur	Variable	←	Texte de l'erreur
ligne	Variable	←	Numéro de ligne
colonne	Variable	←	Numéro de colonne

Description

La commande LIRE ERREUR XSLT retourne dans le paramètre texteErreur la description de la dernière erreur éventuellement rencontrée lors de la transformation XSLT effectuée dans le process courant. Les informations retournées sont fournies par la librairie Xerces.dll.

Les paramètres optionnels ligne et colonne désignent l'emplacement de l'erreur dans le fichier XSL : ils récupèrent respectivement le numéro de la ligne et, dans cette ligne, la position du premier caractère de l'expression à l'origine de l'erreur.

Référence

APPLIQUER TRANSFORMATION XSLT, FIXER PARAMETRE XSLT.

Variables et ensembles système

Si la commande a été correctement exécutée, la variable système OK prend la valeur 1. Si une erreur se produit, elle prend la valeur 0.

SVG Chercher ID element par coordonnees ({*; }objetImage; x; y) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objetImage est un nom d'objet (chaîne) Si omis, objetImage est un champ ou une variable
objetImage	Image	→	Nom d'objet (si * spécifié) ou Champ ou Variable (si * omis)
x	Entier long	→	Coordonnée X en pixels
y	Entier long	→	Coordonnée Y en pixels
Résultat	Chaîne	←	ID de l'élément se trouvant à l'emplacement x,y

Description

La commande SVG Chercher ID element par coordonnees retourne l'ID (attribut "id" ou "xml:id") de l'élément XML situé à l'emplacement défini par les coordonnées (x,y) dans l'image SVG désignée par le paramètre objetImage. Cette commande permet notamment de créer des interfaces graphiques interactives utilisant des objets SVG.

Note : Pour plus d'informations sur le format SVG, reportez-vous à la section Présentation des commandes XML Utilitaires.

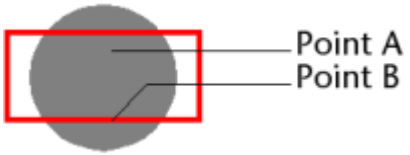
Si vous passez le paramètre optionnel *, vous indiquez que le paramètre objetImage est un nom d'objet (une chaîne). Si vous ne passez pas le paramètre, vous indiquez que le paramètre objetImage est un champ ou une variable. Dans ce cas, vous ne passez pas une chaîne mais une référence de champ ou de variable (champ ou variable objet uniquement).

A noter qu'il n'est pas obligatoire que l'image soit affichée dans un formulaire. Dans ce cas, la syntaxe de type "nom d'objet" n'est pas valide, vous devez passer un nom de champ ou de variable.

Les coordonnées passées dans les paramètres x et y doivent être exprimées en pixels relativement à l'angle supérieur gauche de l'image (0,0). Dans le contexte d'une image affichée dans un formulaire, vous pouvez utiliser les valeurs retournées par les variables système MouseX et MouseY. Ces variables sont mises à jour dans les événements formulaire Sur clic, Sur double clic ainsi que Sur début survol et Sur survol.

Note : Dans le système de coordonnées des images, [x;y] définit toujours le même point, quel que soit le format d'affichage de l'image, hormis pour le format "mosaïque".

Le point pris en compte est le premier point atteint. Par exemple, dans le cas ci-dessous, la commande retournera l'ID du cercle si les coordonnées du point A sont passées et celui du rectangle si les coordonnées du point B sont passées :



Si les coordonnées correspondent à des objets superposés ou composites, la commande retourne l'ID du premier objet disposant d'un attribut ID valide en remontant si nécessaire parmi les éléments parents.

La commande retourne une chaîne vide si :
• la racine est atteinte sans qu'un attribut "id" ait été trouvé,
• le point de coordonnées n'appartient à aucun objet,
• l'attribut "id" est une chaîne vide.

Variables et ensembles système

Si objetImage ne contient pas une image SVG valide, la commande retourne une chaîne vide et la variable système OK prend la valeur 0. Sinon, si la commande a été exécutée correctement, la variable système OK prend la valeur 1.

SVG EXPORTER VERS IMAGE (refElément; vVarImage{; typeExport})

Paramètre	Type		Description
refElément	Chaîne	→	Référence d'élément XML racine
vVarImage	Image	→	Variable image devant recevoir l'arbre XML (image SVG)
typeExport	Entier long	→	0=Ne pas stocker la source de données 1=Copier la source de données, 2 (défaut) = Prendre possession de la source de données

Description

La commande SVG EXPORTER VERS IMAGE permet de sauvegarder dans la variable ou le champ image désigné(e) par le paramètre vVarImage une image au format SVG contenue dans un arbre XML.

Note : Pour plus d'informations sur le format SVG, reportez-vous à la section Présentation des commandes XML Utilitaires.

Passez dans refElément la référence de l'élément XML racine contenant l'image SVG.

Passez dans vVarImage le nom de la variable image ou du champ image 4D devant contenir l'image SVG. L'image est exportée dans son format natif (description XML) et est dessinée via le moteur de rendu SVG au moment de l'affichage.

Le paramètre facultatif typeExport vous permet de définir la manière dont la source de données XML doit être prise en charge par la commande. Vous pouvez passer dans ce paramètre une des constantes suivantes, placées dans le thème "XML" :
• Lire source données XML (0) : 4D lit uniquement la source de données XML, elle n'est pas conservée avec l'image. Ce paramétrage accélère sensiblement l'exécution de la commande, toutefois l'arbre DOM n'étant pas conservé, il ne sera pas possible de stocker ni d'exporter l'image.
• Copier source données XML (1) : 4D conserve une copie de l'arbre DOM avec l'image, ce qui permet de la sauvegarder dans un champ image de la base de données et de la réafficher ou de l'exporter à tout moment.
• Posséder source données XML (2) : 4D exporte l'arbre DOM avec l'image. L'image pourra être stockée ou exportée et l'exécution de la commande est rapide. Toutefois, la référence XML refElément n'est alors plus utilisable par les autres commandes 4D. Ce mode d'exportation est utilisé par défaut si le paramètre typeExport est omis.

Exemple

L'exemple suivant permet d'afficher "Hello World" dans une image 4D :

```
C_IMAGE(vImage)
$svg:=DOM Creer ref XML("svg";"http://www.w3.org/2000/svg")
$ref:=DOM Creer element XML($svg;"text";"font-size";26;"fill";"red")
DOM ECRIRE VALEUR ELEMENT XML($ref;"Hello World")
SVG EXPORTER VERS IMAGE($svg;vImage;Copier source données XML)
DOM FERMER XML($svg)
```

Référence

DOM EXPORTER VERS FICHIER, DOM EXPORTER VERS VARIABLE.

68

Zone Web

Les commandes de ce thème sont dédiées à la gestion programmée des objets de formulaire de type Zone Web (*Web Area*).

Les zones Web peuvent afficher tout type de contenu Web à l'intérieur même de votre environnement 4D : pages HTML au contenu statique ou dynamique, fichiers, images, Javascript, Flash, PDF... et même des documents MS Office (sous Windows lorsque la suite MS Office est installée). L'image suivante montre une zone Web incluse dans un formulaire et affichant une page HTML :

Outre les commandes du thème Zone Web, plusieurs actions standard et des événements formulaires dédiés permettent au développeur de contrôler le fonctionnement des zones Web. Des variables spécifiques prennent en charge l'interaction entre la zone Web et l'environnement 4D. Ces outils vous permettent ainsi de développer un navigateur Web basique dans vos formulaires.

Créer et adresser une zone Web

La création d'une zone Web s'effectue à l'aide d'une variante du bouton Zone de plug-in/Sous-formulaire de la barre d'objets de l'éditeur de formulaires de 4D (pour plus d'information, reportez-vous au manuel *Mode Développement*).

Comme les autres objets dynamiques de formulaire, une zone Web dispose d'un nom d'objet et d'un nom de variable, vous permettant de l'adresser par programmation. La variable standard associée à l'objet zone Web est de type Texte. Vous pouvez en particulier utiliser les commandes CHOIX VISIBLE et DEPLACER OBJET avec les zones Web.

Note : La variable Texte associée à la zone Web ne contient pas de référence et ne peut donc pas être passée en tant que paramètre à une méthode. Par exemple, pour une zone Web nommée MaZone, le code suivant ne peut pas être utilisé :

```
Mamethode(MaZone)
```

Code de *Mamethode* :

```
WA ACTUALISER URL($1) `Ne fonctionne pas
```

Pour ce type de programmation, vous devez utiliser des pointeurs :

```
Mamethode(->MaZone)
```

Code de *Mamethode* :

```
WA ACTUALISER URL($1->) `Fonctionne
```

Mode compositing (Mac OS)

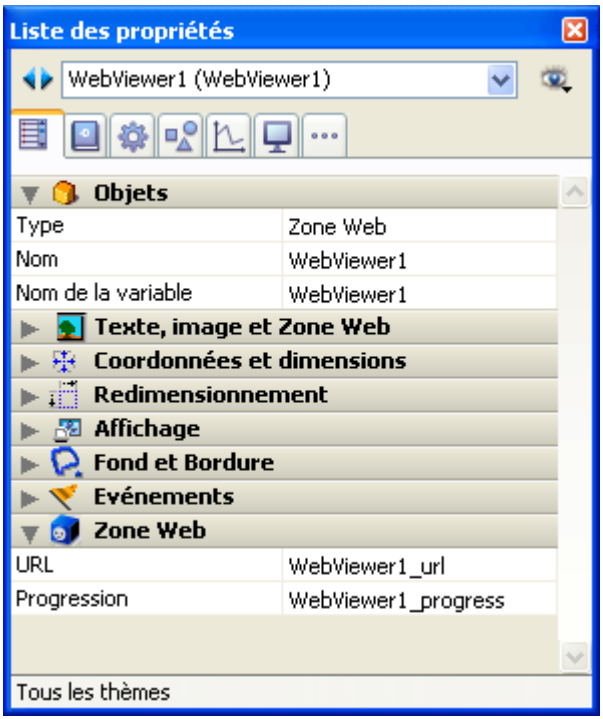
Pour pouvoir être affichées sous Mac OS, les zones Web doivent être incluses dans des fenêtres dessinées en "mode compositing". Ce mode interne de gestion des fenêtres sous Mac OS n'est pas utilisé dans toutes les fenêtres de 4D.
Dans 4D v11 SQL, les fenêtres dessinées en "mode compositing" sont :
• les fenêtres générées par les commandes Creer fenetre et Creer fenetre formulaire ayant le type Mode compositing (constante de valeur 4096) ;
• en mode Développement, les fenêtres affichant un formulaire projet.

Note : Certains objets d'ancienne architecture ne sont pas compatibles avec le mode compositing (par exemple les zones 4D Chart). S'ils sont affichés dans des fenêtres en mode compositing, ces objets ne fonctionneront pas.

Gestion des variables associées

Outre la variable objet standard (cf. paragraphe précédent), deux variables spécifiques sont automatiquement associées à chaque zone Web :
• la variable "URL"
• la variable "Progression du chargement"
Par défaut, ces variables sont nommées respectivement *nomZone*_url et *nomZone*_progress.

Vous pouvez modifier ces noms comme vous le souhaitez. Les variables sont accessibles dans la Liste des propriétés :



Variable URL
La variable "URL" est de type chaîne. Elle contient l'URL chargé ou en cours de chargement par la zone Web associée.
L'association entre la variable et la zone Web s'effectue dans les deux sens :
• Si l'utilisateur affecte un nouvel URL à la variable, l'URL est automatiquement chargé par la zone Web.
• Toute navigation effectuée à l'intérieur de la zone Web met automatiquement à jour le contenu de la variable.
Schématiquement, cette variable fonctionne comme la zone d'adresse d'un navigateur Web. Vous pouvez la représenter par une zone de texte située au-dessus de la zone Web.

Variable URL et commande WA OUVRIR URL
La variable URL produit les mêmes effets que la commande WA OUVRIR URL. Les différences suivantes sont toutefois à noter :
- pour les accès aux documents, la variable accepte uniquement des URLs conformes aux RFC ("file://c:/Mon%20Doc") et non les chemins d'accès système ("c:\MonDoc"). La commande WA OUVRIR URL accepte les deux notations.
- si la variable URL contient une chaîne vide, la zone Web ne tente pas de charger l'URL. La commande WA OUVRIR URL génère une erreur dans ce cas.

- si la variable URL ne contient pas de protocole (http, mailto, file, etc.), la zone Web ajoute "http://", ce qui n'est pas le cas pour la commande WA OUVRIR URL.
- lorsque la zone Web n'est pas affichée dans le formulaire (lorsqu'elle se trouve sur une autre page du formulaire), l'exécution de la commande WA OUVRIR URL est sans effet tandis que la valorisation de la variable URL permet de mettre à jour l'URL courant.

Variable Progression du chargement
La variable "Progression du chargement" est de type Entier long. Elle contient une valeur entre 0 et 100, représentant le pourcentage du chargement complet de la page affichée dans la zone Web.
La variable est mise à jour automatiquement par 4D. Il n'est pas possible de la modifier manuellement.

Evénements formulaire

Des événements formulaire spécifiques sont destinés à la gestion programmée des zones Web, concernant notamment l'activation des liens :
- Sur début chargement URL
- Sur chargement ressource URL
- Sur fin chargement URL
- Sur erreur chargement URL
- Sur filtrage URL
- Sur ouverture lien externe
- Sur refus ouverture fenêtre

En outre, les zones Web prennent en charge les événements formulaire génériques suivants :
- Sur chargement
- Sur libération
- Sur gain focus
- Sur perte focus
- Sur glisser
- Sur déposer
- Sur début glisser

Pour plus d'informations sur ces événements, reportez-vous à la description de la commande Evenement formulaire.

Notes d'utilisation des zones Web

Interface utilisateur
Lors de l'exécution du formulaire, l'utilisateur dispose des fonctions d'interface standard des navigateurs dans la zone Web, ce qui lui permet d'interagir avec les autres zones du formulaire :
• *commandes du menu* ***Edition*** : lorsque la zone Web a le focus, les commandes du menu **Edition** permettent d'effectuer les actions de copier, coller, tout sélectionner, etc., en fonction de la sélection.

• *menu contextuel* : il est possible d'associer un menu contextuel standard à la zone Web via la Liste des propriétés. L'affichage de ce menu peut également être contrôlé via la commande WA FIXER PREFERENCE).
• *glisser-déposer* : l'utilisateur peut effectuer des glisser-déposer de textes, d'images ou de documents à l'intérieur d'une zone Web ou entre une zone Web et les objets des formulaires 4D, en fonction des propriétés des objets 4D.

Documents MS Office (Windows)
Sous Windows, les zones Web peuvent prendre en charge l'affichage et la modification de documents Microsoft Office (lorsque Microsoft Office est installé sur le poste). En particulier, les documents Word, Excel et Powerpoint (extensions .doc, .xls et .ppt) peuvent être traités. Le format XML MS Office est également pris en charge.

Note : MS Office 2007 ne permet pas par défaut l'affichage de documents dans un navigateur Web, ils sont toujours ouverts dans une nouvelle fenêtre. Vous pouvez modifier ce fonctionnement à l'aide des instructions fournies à cette adresse :
http://support.microsoft.com/kb/162059/en-us
Sous Windows, les zones Web permettent d'afficher des dossiers locaux ou externes via le protocole ftp:// ou via les chemins réseau (\\monserveur\monvolume).

Conflit Zone Web et serveur Web (Windows)
Sous Windows, il est déconseillé d'accéder via une zone Web au serveur Web de l'application 4D contenant la zone car cette configuration peut provoquer un conflit paralysant l'application. Bien entendu, un 4D distant peut accéder au serveur Web du 4D Server, mais pas à son propre serveur Web.

Insertion du protocole (Mac OS)
Les URLs manipulés par programmation dans les zones Web sous Mac OS doivent débuter par le protocole. Par exemple, vous devez passer la chaîne "http://www.monsite.fr" et non uniquement "www.monsite.fr".

WA ACTUALISER URL

WA ACTUALISER URL ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande WA ACTUALISER URL provoque le rechargement de l'URL courant affiché dans la zone Web désignée par les paramètres * et objet.

Référence

WA ARRETER CHARGEMENT URL.

WA AGRANDIR TEXTE PAGE

WA AGRANDIR TEXTE PAGE ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande WA AGRANDIR TEXTE PAGE augmente la taille du texte affiché dans la zone Web désignée par les paramètres * et objet.

Sous Mac OS, la portée de cette commande est la session 4D : le paramétrage effectué n'est pas conservé après la fermeture de l'application 4D.

Sous Windows, la portée de cette commande est globale : le paramétrage est conservé après la fermeture de l'application 4D.

Référence

WA REDUIRE TEXTE PAGE.

WA ARRETER CHARGEMENT URL

WA ARRETER CHARGEMENT URL ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande WA ARRETER CHARGEMENT URL stoppe le chargement des ressources de l'URL courant de la zone Web désignée par les paramètres * et objet.

Référence

WA ACTUALISER URL.

WA Creer menu historique URL

WA Creer menu historique URL ({*; }objet{; direction}) → RefMenu

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
direction	Entier	→	0 ou omis=Liste des URLs précédents, 1=Liste des URLs suivants
Résultat	RefMenu	←	Référence du menu

Description

La commande WA Creer menu historique URL crée et remplit un menu pouvant être utilisé directement pour la navigation parmi les URLs visités au cours de la session dans la zone Web désignée par les paramètres * et objet. Elle permet de construire une interface de navigation personnalisée.

Les informations fournies concernent la session, c'est-à-dire la navigation effectuée dans une même zone Web tant que le formulaire n'a pas été refermé.

Passez dans direction une valeur indiquant la liste à récupérer. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème "Zone Web" :

Constante	Type	Valeur
wa URLs précédents	Entier long	0
wa URLs suivants	Entier long	1

Si vous omettez le paramètre direction, la valeur 0 est utilisée.

Une fois le menu généré, vous pouvez l'afficher via la commande de 4D Pop up menu dynamique et le manipuler via les commandes standard de gestion des menus de 4D. La référence RefLigne (de type chaîne) retournée par la commande Pop up menu dynamique contient l'URL de la page visitée (voir exemple).

Appelez la commande EFFACER MENU pour supprimer un menu historique d'URL lorsqu'il est devenu inutile.

Exemple

Le code suivant pourrait être associé à un bouton 3D avec pop up menu libellé "Précédent" :

```
Au cas ou
: (Evenement formulaire=Sur clic)  `Clic simple
        WA OUVRIR URL PRECEDENT(WA_zone)
: (Evenement formulaire=Sur clic flèche) `Clic sur la flèche -> affichage du pop up
        `Créer un menu historique précédent
    $Menu:=WA Creer menu historique URL(WA_zone;wa URLs précédents)
            `Afficher ce menu dans un pop up
    $URL:=Pop up menu dynamique($Menu)
    Si ($URL#"") `Si une ligne est sélectionnée
        WA OUVRIR URL(WA_zone;$URL)  `Ouvrir la page Web
    Fin de si
    EFFACER MENU($Menu) `Effacer le menu pour libérer la mémoire
Fin de cas
```

Référence

EFFACER MENU, Pop up menu dynamique, WA LIRE HISTORIQUE URL.

WA EXECUTER FONCTION JAVASCRIPT ({*; }objet; fonctionJS; résultat|*{; param}{; param2; ...; paramN})

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
fonctionJS	Chaîne	→	Nom de la fonction JavaScript à exécuter
résultat|*	Variable	→	* pour une fonction sans résultat ou
		←	Résultat de la fonction (si attendu)
param	Chaîne	→	Paramètre(s) à passer à la fonction

Description

La commande WA EXECUTER FONCTION JAVASCRIPT exécute dans la zone Web désignée par les paramètres * et objet la fonction JavaScript fonctionJS et retourne facultativement son résultat dans le paramètre résultat.
Si la fonction ne retourne pas de résultat, passez * dans le paramètre résultat.

Vous pouvez passer dans param une ou plusieurs chaîne(s) contenant les paramètres de la fonction.

Exemple

Appel d'une fonction JavaScript avec 3 paramètres :

```
$JavaScriptFunction:="TheFunctionToBeExecuted"
$Param1:="10"
$Param2:="true"
$Param3:="1,000.2" `notez "," comme séparateur de milliers et "." comme séparateur décimal
WA EXECUTER FONCTION JAVASCRIPT(MaZoneW; $JavaScriptFunction;$Result; $Param1; $Param2; $Param3)
```

Référence

WA Executer JavaScript.

WA Executer JavaScript ({*; }objet; codeJS) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
codeJS	Chaîne	→	Code JavaScript
Résultat	Chaîne	←	Résultat de l'exécution

Description

La commande WA Executer JavaScript exécute dans la zone Web désignée par les paramètres * et objet le code JavaScript passé dans codeJS.

• Sous Mac OS, la commande retourne le résultat.
• Sous Windows, la commande retourne une chaîne vide. Utilisez la commande WA EXECUTER FONCTION JAVASCRIPT.

Exemple

Cet exemple de code JavaScript provoque l'affichage de l'url précédent :

```
$résultat:=WA Executer JavaScript(MaZoneW;"history.back()")
```

Référence

WA EXECUTER FONCTION JAVASCRIPT.

WA FIXER CONTENU PAGE

WA FIXER CONTENU PAGE ({*; }objet; contenu; baseURL)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
contenu	Chaîne	→	Code HTML source
baseURL	Chaîne	→	URL pour les références relatives (Mac OS)

Description

La commande WA FIXER CONTENU PAGE remplace la page affichée dans la zone Web désignée par les paramètres * et objet par le code HTML passé dans le paramètre contenu.

Le paramètre baseURL permet de définir sous Mac OS un URL de base qui sera ajouté devant les liens relatifs éventuellement présents dans la page.
Sous Windows, ce paramètre est sans effet, l'URL de base est indéfini. Il n'est donc pas possible d'utiliser des références relatives sur cette plate-forme.

Note : Sous Windows, il est impératif qu'une page ait déjà été chargée dans la zone Web avant que cette commande puisse être appelée. Si nécessaire, vous pouvez passer l'URL "about:blank" afin de charger une page blanche.

Exemple

Affichage de la phrase "Hello world !" et définition d'un URL de base "file:///" (Mac OS uniquement) :

```
WA FIXER CONTENU PAGE(MaZoneW;"<html><body><h1>Hello World!</h1></body>
                                                    </html>";"file:///")
```

Référence

WA Lire contenu page.

WA FIXER FILTRES LIENS EXTERNES ({*; }objet; tabFiltres; tabAutorisRefus)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	→	Tableau de filtres
tabAutorisRefus	Tableau Booléen	→	Tableau autoriser-refuser

Description

La commande WA FIXER FILTRES LIENS EXTERNES permet de mettre en place un ou plusieurs filtre(s) de liens externes pour la zone Web désignée par les paramètres * et objet. Les filtres de liens externes déterminent si un URL associé à la page courante via un lien doit être ouvert dans la zone Web ou dans le navigateur Web par défaut de la machine.

Lorsque l'utilisateur clique sur un lien dans la page courante, 4D consulte la liste des filtres de liens externes afin de vérifier si l'URL demandé doit être ouvert dans le navigateur de la machine. Si c'est le cas, la page correspondant à l'URL est affichée dans le navigateur Web et l'événement formulaire Sur ouverture lien externe est généré (cf. commande Evenement formulaire). Sinon (fonctionnement par défaut), la page correspondant à l'URL est affichée dans la zone Web. L'évaluation de l'URL est basée sur le contenu des tableaux tabFiltres et tabAutorisRefus.

Les tableaux tabFiltres et tabAutorisRefus doivent être synchronisés.
• Chaque ligne du tableau tabFiltres doit contenir un URL devant être filtré. Vous pouvez utiliser le * comme joker de remplacement pour un ou plusieurs caractère(s).
• Chaque ligne correspondante dans le tableau tabAutorisRefus doit contenur un booléen indiquant si l'URL doit être ouvert dans la zone Web (Vrai) ou dans le navigateur Web (Faux).

En cas de contradiction au niveau des paramétrages (autorisation et refus d'un même URL), le paramétrage pris en compte est le dernier.

Pour désactiver le filtrage des URLs, appelez la commande en lui passant des tableaux vides ou en passant respectivement les valeurs "*" et Vrai dans la dernière ligne des tableaux tabFiltres et tabAutorisRefus.

Important : Le filtrage établi par la commande WA FIXER FILTRES URL est pris en compte avant celui de WA FIXER FILTRES LIENS EXTERNES. Cela signifie que si un URL est refusé à cause d'un filtre de la commande WA FIXER FILTRES URL, il ne pourra pas être ouvert dans un navigateur même s'il est explicitement défini par la commande WA FIXER FILTRES LIENS EXTERNES (cf. exemple 2).

Exemples

(1) Cet exemple provoquera l'ouverture de sites dans des navigateurs externes :

```
TABLEAU ALPHA(0;$filters;0)
TABLEAU BOOLEEN($AllowDeny;0)

AJOUTER A TABLEAU($filters;"*www.google.*") `Sélectionner "google"
AJOUTER A TABLEAU($AllowDeny;Faux) `Faux : ce lien sera ouvert dans un navigateur externe
AJOUTER A TABLEAU($filters;"*www.apple.*")
AJOUTER A TABLEAU($AllowDeny;Faux) `Faux : ce lien sera ouvert dans un navigateur externe
WA FIXER FILTRES LIENS EXTERNES(MaZoneW;$filters;$AllowDeny)
```

(2) Cet exemple combine des filtrages de sites et de liens externes :

```
TABLEAU ALPHA(0;$filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*www.google.*") `Sélectionner "google"
AJOUTER A TABLEAU($AllowDeny;Faux) `Interdire ce lien
WA FIXER FILTRES URL(MaZoneW;$filters;$AllowDeny)

TABLEAU ALPHA(0;$filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*www.google.*") `Sélectionner "google"
AJOUTER A TABLEAU($AllowDeny;Faux)
   `Faux : ce lien devrait être ouvert dans un navigateur externe, mais ce paramétrage est
   `sans effet car le lien sera bloqué du fait du filtrage d'URL.
WA FIXER FILTRES LIENS EXTERNES(MaZoneW;$filters;$AllowDeny)
```

Référence

WA FIXER FILTRES URL, WA LIRE FILTRES LIENS EXTERNES.

WA FIXER FILTRES URL ({*; }objet; tabFiltres; tabAutorisRefus)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	→	Tableau de filtres
tabAutorisRefus	Tableau Booléen	→	Tableau autoriser-refuser

Description

La commande WA FIXER FILTRES URL permet de mettre en place un ou plusieurs filtre(s) pour la zone Web désignée par les paramètres * et objet.

Avant le chargement de toute page demandée par l'utilisateur, 4D consulte la liste des filtres afin de vérifier si l'URL cible est autorisé ou non. L'évaluation de l'URL est basée sur le contenu des tableaux tabFiltres et tabAutorisRefus s'ils ont été définis.

Si l'URL demandé n'est pas autorisé, il n'est pas chargé et l'événement formulaire Sur filtrage URL est généré (cf. commande Evenement formulaire).

Les tableaux tabFiltres et tabAutorisRefus doivent être synchronisés.
• Chaque ligne du tableau tabFiltres doit contenir un URL devant être filtré. Vous pouvez utiliser le * comme joker de remplacement pour un ou plusieurs caractère(s).
• Chaque ligne correspondante dans le tableau tabAutorisRefus doit contenur un booléen indiquant si l'URL doit être autorisé (Vrai) ou refusé (Faux).

En cas de contradiction au niveau des paramétrages (autorisation et refus d'un même URL), le paramétrage pris en compte est le dernier.

Pour désactiver le filtrage des URLs, appelez la commande en lui passant des tableaux vides ou en passant respectivement les valeurs "*" et Vrai dans la dernière ligne des tableaux tabFiltres et tabAutorisRefus.

Une fois la commande exécutée, les filtres deviennent une propriété de la zone Web. Si les tableaux tabFiltres et tabAutorisRefus sont supprimés ou réinitialisés, les filtres restent actifs tant que la commande n'a pas été exécutée à nouveau. Pour connaître les filtres actifs pour une zone, vous devez utiliser la commande WA LIRE FILTRES URL.

Important : Le filtrage des URLs effectué par cette commande s'applique uniquement à la variable "URL" associée à la zone Web (variable généralement saisissable et affichée dans le formulaire). Le filtrage ne s'applique pas à la commande WA OUVRIR URL ni aux autres commandes de navigation.

Exemples

(1) Vous souhaitez interdire l'accès à tous les sites web .org, .net et .fr :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*.org")
AJOUTER A TABLEAU($AllowDeny;Faux)
AJOUTER A TABLEAU($filters;"*.net")
AJOUTER A TABLEAU($AllowDeny;Faux)
AJOUTER A TABLEAU($filters;"*.fr")
AJOUTER A TABLEAU($AllowDeny;Faux)
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

(2) Vous souhaitez interdire l'accès à tous les sites web sauf les sites russes (.ru) :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Faux) `Tout interdire
AJOUTER A TABLEAU($filters;"www.*.ru") `Sélectionner *.ru
AJOUTER A TABLEAU($AllowDeny;Vrai) `Autoriser
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

(3) Vous souhaitez donner accès aux sites Web 4D uniquement (.com, .fr, .es, etc.) :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Faux) `Tout interdire
AJOUTER A TABLEAU($filters;"www.4D.*") `Sélectionner 4d.fr, 4d.com...
AJOUTER A TABLEAU($AllowDeny;Vrai) `Autoriser
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

(4) Vous souhaitez autoriser l'accès local à la documentation uniquement (située dans le dossier C://doc) :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Faux) `Tout interdire
AJOUTER A TABLEAU($filters;"file://C:/doc/*") `Sélectionner le chemin file:// autorisé
AJOUTER A TABLEAU($AllowDeny;Vrai)-> Autoriser
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

(5) Vous souhaitez autoriser tous les sites sauf un, par exemple celui d'Elcaro :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*")
AJOUTER A TABLEAU($AllowDeny;Vrai) `Tout autoriser
AJOUTER A TABLEAU($filters;"*elcaro*") `Interdire tout ce qui contient elcaro
AJOUTER A TABLEAU($AllowDeny;Faux)
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

(6) Vous souhaitez interdire des adresses IP spécifiques :

```
TABLEAU TEXTE($filters;0)
TABLEAU BOOLEEN($AllowDeny;0)
AJOUTER A TABLEAU($filters;"*") `Tout sélectionner
AJOUTER A TABLEAU($AllowDeny;Vrai) `Tout autoriser
AJOUTER A TABLEAU($filters;"86.83.*") `Sélectionner les IP débutant par 86.83.
AJOUTER A TABLEAU($AllowDeny;Faux) `Interdire
AJOUTER A TABLEAU($filters;"86.1*") `Sélectionner les IP débutant par 86.1 (86.10,
                                                                    86.135 etc.)
AJOUTER A TABLEAU($AllowDeny;Faux) `Interdire
   `A noter que l'adresse IP d'un domaine peut varier
WA FIXER FILTRES URL(MyWArea;$filters;$AllowDeny)
```

Référence

WA FIXER FILTRES LIENS EXTERNES, WA LIRE FILTRES URL.

WA FIXER PREFERENCE

WA FIXER PREFERENCE ({*; }objet; sélecteur; valeur)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
sélecteur	Entier long	→	Préférence à modifier
valeur	Booléen	→	Valeur de la préférence (Vrai = autorisé, Faux = non autorisé)

Description

La commande WA FIXER PREFERENCE permet de fixer différentes préférences pour la zone Web désignée par les paramètres * et objet.

Passez dans le paramètre sélecteur la préférence à modifier et dans valeur la valeur à lui attribuer. Vous pouvez passer dans sélecteur l'une des constantes suivantes, placées dans le thème "Zone Web" :

Constante	Type	Valeur
wa autoriser applets Java	Entier long	1
wa autoriser JavaScript	Entier long	2
wa autoriser plugins	Entier long	3
wa autoriser menu contextuel	Entier long	4

Pour chaque préférence, passez Vrai dans valeur pour l'activer et Faux pour l'inactiver.

Voici la signification des sélecteurs :
- wa autoriser applets Java : permet d'autoriser l'exécution d'applets Java dans la zone Web.
- wa autoriser JavaScript : permet d'autoriser l'exécution de code JavaScript dans la zone Web.
- wa autoriser plugins : permet d'autoriser l'installation de plug-ins dans la zone Web.
- wa autoriser menu contextuel : permet d'autoriser l'affichage du menu contextuel standard dans la zone Web.

Référence

WA LIRE PREFERENCE.

WA Lire contenu page

WA Lire contenu page ({*; }objet) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	←	Code HTML source

Description

La commande WA Lire contenu page retourne le code HTML de la page courante ou en cours d'affichage dans la zone Web désignée par les paramètres * et objet.

Cette commande retourne une chaîne vide si le contenu de la page courante n'est pas disponible.

Référence

WA FIXER CONTENU PAGE.

WA Lire dernier URL filtre

WA Lire dernier URL filtre ({*; }objet) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	←	Dernier URL filtré

Description

La commande WA Lire dernier URL filtre retourne le dernier URL ayant été filtré dans la zone Web désignée par les paramètres * et objet.

L'URL peut avoir été filtré pour l'une des raisons suivantes :
• l'URL est interdit à cause d'un filtre (commande WA FIXER FILTRES URL),
• le lien est ouvert dans le navigateur par défaut (commande WA FIXER FILTRES LIENS EXTERNES),
• l'URL tentait d'ouvrir une fenêtre pop up.

Il est judicieux d'appeler cette commande dans le contexte des événements formulaire Sur filtrage URL, Sur ouverture lien externe et Sur refus ouverture fenêtre afin de connaître l'URL filtré. Pour plus d'informations, reportez-vous à la description de la commande Evenement formulaire.

Référence

WA FIXER FILTRES LIENS EXTERNES, WA FIXER FILTRES URL, WA LIRE FILTRES LIENS EXTERNES, WA LIRE FILTRES URL.

WA LIRE DERNIERE ERREUR URL ({*; }objet; url; description; codeErreur)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
url	Chaîne	←	URL à l'origine de l'erreur
description	Chaîne	←	Description de l'erreur (Mac OS)
codeErreur	Entier long	←	Code d'erreur

Description

La commande WA LIRE DERNIERE ERREUR URL vous permet de récupérer plusieurs informations relatives à la dernière erreur ayant eu lieu dans la zone Web désignée par les paramètres * et objet.
Ces informations sont retournées dans trois variables :
• url : l'URL ayant provoqué l'erreur.
• description (Mac OS uniquement) : un texte décrivant l'erreur (si disponible). S'il n'est pas possible d'associer un texte à l'erreur, une chaîne vide est retournée. Sous Windows, ce paramètre est toujours retourné vide.
• codeErreur : code de l'erreur.
- Si le code est >=400, il s'agit d'une erreur liée au protocole HTTP. Pour plus d'informations sur ce type d'erreur, reportez-vous à l'adresse http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- Sinon, il s'agit d'une erreur retournée par le WebKit (Mac OS) ou ActiveX (Windows).

Il est judicieux d'appeler cette commande dans le cadre de l'événement formulaire Sur erreur chargement URL afin de connaître la cause de l'erreur qui vient de se produire. Pour plus d'informations, reportez-vous à la description de la commande Evenement formulaire.

Référence

Evenement formulaire.

WA LIRE FILTRES LIENS EXTERNES

WA LIRE FILTRES LIENS EXTERNES ({*; }objet; tabFiltres; tabAutorisRefus)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	←	Tableau de filtres
tabAutorisRefus	Tableau Booléen	←	Tableau autoriser-refuser

Description

La commande WA LIRE FILTRES LIENS EXTERNES retourne dans les tableaux tabFiltres et tabAutorisRefus les filtres de liens externes de la zone Web désignée par les paramètres * et objet. Si aucun filtre n'est actif, les tableaux sont retournés vides.

Les filtres sont installés par la commande WA FIXER FILTRES LIENS EXTERNES. Si les tableaux ont été réinitialisés au cours de la session, la commande WA LIRE FILTRES LIENS EXTERNES vous permet de connaître le paramétrage courant.

Référence

WA FIXER FILTRES LIENS EXTERNES, WA LIRE FILTRES URL.

WA LIRE FILTRES URL ({*; }objet; tabFiltres; tabAutorisRefus)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabFiltres	Tableau chaîne	←	Tableau de filtres
tabAutorisRefus	Tableau Booléen	←	Tableau autoriser-refuser

Description

La commande WA LIRE FILTRES URL retourne dans les tableaux tabFiltres et tabAutorisRefus les filtres actifs dans la zone Web désignée par les paramètres * et objet. Si aucun filtre n'est actif, les tableaux sont retournés vides.

Les filtres sont installés par la commande WA FIXER FILTRES URL. Si les tableaux ont été réinitialisés au cours de la session, la commande WA LIRE FILTRES URL vous permet de connaître le paramétrage courant.

Référence

WA FIXER FILTRES URL, WA LIRE FILTRES LIENS EXTERNES.

WA LIRE HISTORIQUE URL

WA LIRE HISTORIQUE URL ({*; }objet; tabsUrls{; direction{; tabTitres}})

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
tabsUrls	Tab Alpha	←	Tableau des URLs visités
direction	Entier	→	0 ou omis=Liste des URLs précédents, 1=Liste des URLs suivants
tabTitres	Tab Alpha	←	Tableau des titres de fenêtres

Description

La commande WA LIRE HISTORIQUE URL retourne un ou deux tableaux contenant les URLs visités au cours de la session dans la zone Web désignée par les paramètres * et objet. Elle permet de construire une interface de navigation personnalisée.

Les informations fournies concernent la session, c'est-à-dire la navigation effectuée dans une même zone Web tant que le formulaire n'a pas été refermé.

Le tableau tabUrls est rempli avec la liste des URLs visités. En fonction de la valeur du paramètre direction (s'il est passé), le tableau récupère la liste des URLs précédents (fonctionnement par défaut) ou la liste des URLs suivants. Ces listes correspondent au contenu des boutons standard Précédent et Suivant des navigateurs.

Les URLs sont classés par ordre chronologique.

Passez dans direction une valeur indiquant la liste à récupérer. Vous pouvez utiliser l'une des constantes suivantes, placées dans le thème "Zone Web" :

Constante	Type	Valeur
wa URLs précédents	Entier long	0
wa URLs suivants	Entier long	1

Si vous omettez le paramètre direction, la valeur 0 est utilisée.

S'il est passé, le paramètre tabTitres contient la liste des noms de fenêtres associés aux URLs. Ce tableau est synchronisé avec le tableau tabUrls.

Référence

WA Creer menu historique URL.

WA LIRE PREFERENCE

WA LIRE PREFERENCE ({*; }objet; sélecteur; valeur)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
sélecteur	Entier long	→	Préférence à lire
valeur	Variable	←	Valeur courante de la préférence

Description

La commande WA LIRE PREFERENCE permet de lire la valeur courante d'une préférence dans la zone Web désignée par les paramètres * et objet.

Passez dans le paramètre sélecteur la préférence à lire. Vous pouvez passer l'une des constantes suivantes, placées dans le thème "Zone Web" :

Constante	Type	Valeur
wa autoriser applets Java	Entier long	1
wa autoriser JavaScript	Entier long	2
wa autoriser plugins	Entier long	3
wa autoriser menu contextuel	Entier long	4

Pour plus d'informations sur ces préférences, reportez-vous à la description de la commande WA FIXER PREFERENCE.

Passez dans le paramètre valeur une variable devant recevoir la valeur courante de la préférence. Le type de la variable dépend de la préférence. Dans la version actuelle de 4D v11 SQL, la variable valeur est toujours de type booléen : elle contient Vrai si la préférence est active et Faux sinon.

Référence

WA FIXER PREFERENCE.

WA Lire titre page

WA Lire titre page ({*; }objet) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	←	Titre de la page courante

Description

La commande WA Lire titre page retourne le titre de la page courante ou en cours d'affichage dans la zone Web désignée par les paramètres * et objet. Le titre correspond à la balise HTML "Title".

Cette commande retourne une chaîne vide s'il n'y a pas de titre disponible à l'URL courant.

Référence

WA Lire contenu page.

WA Lire URL courant

WA Lire URL courant({*; } objet) → Chaîne

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Chaîne	←	URL actuellement chargé dans la zone Web

Description

La commande WA Lire URL courant retourne l'adresse URL de la page affichée dans la zone Web désignée par les paramètres * et objet.

Si l'URL courant n'est pas disponible, la commande retourne une chaîne vide.

Si la page Web est entièrement chargée, la valeur retournée par la fonction est identique à celle de la variable "URL" associée à la zone Web. Si la page est en cours de chargement, les deux valeurs seront différentes : la fonction retourne l'URL entièrement chargé et la variable contient l'URL en cours de chargement.

Exemple

La page affichée est l'URL "www.apple.com" et la page "www.4d.com" est en cours de chargement :

```
$url:=WA Lire URL courant(MaZoneW) `retourne "http://www.apple.com"
    `La variable URL associée contient "http://www.4d.com"
```

Référence

WA OUVRIR URL.

WA OUVRIR URL ({*; }objet; url)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
url	Chaîne	→	URL à charger dans la zone Web

Description

La commande WA OUVRIR URL charge dans la zone Web désignée par les paramètres * et objet l'URL passé dans le paramètre url.

Si une chaîne vide est passée dans url, la commande WA OUVRIR URL ne fait rien et aucune erreur n'est générée. Pour charger une page vide dans la zone Web, passez la chaîne "about:blank" dans url.

Comme la commande OUVRIR URL WEB, WA OUVRIR URL accepte plusieurs types de syntaxes dans le paramètre url pour désigner les fichiers :
• syntaxe posix : "file://c:/Mon%20Fichier"
• syntaxe système : "c:\MonDossier\MonFichier" (Windows) ou "MonDisque:MonDossier:MonFichier" (Mac OS).

Cette commande a le même effet que la modification de la valeur de la variable "URL" associée à la zone. Par exemple, si la variable de la zone est nommée *MaZoneW_url* :

```
MaZoneW_url:="http://www.4d.com/"
```

équivaut à :

```
WA OUVRIR URL(MaZoneW;"http://www.4d.com/")
```

Référence

OUVRIR URL WEB, WA OUVRIR URL PRECEDENT, WA OUVRIR URL SUIVANT.

WA OUVRIR URL PRECEDENT

WA OUVRIR URL PRECEDENT ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande WA OUVRIR URL PRECEDENT charge dans la zone Web désignée par les paramètres * et objet l'URL précédent dans la séquence d'URLs ouverts.

S'il n'y a pas d'URL précédent, la commande ne fait rien. Vous pouvez tester la disponibilité d'un URL précédent à l'aide de la commande WA URL precedent disponible.

Référence

WA OUVRIR URL, WA OUVRIR URL SUIVANT.

WA OUVRIR URL SUIVANT ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande WA OUVRIR URL SUIVANT charge dans la zone Web désignée par les paramètres * et objet l'URL suivant dans la séquence d'URLs ouverts.

S'il n'y a pas d'URL suivant (c'est-à-dire si l'utilisateur n'a jamais effectué de retour à l'URL précédent), la commande ne fait rien. Vous pouvez tester la disponibilité d'un URL suivant à l'aide de la commande WA URL suivant disponible.

Référence

WA OUVRIR URL, WA OUVRIR URL PRECEDENT.

WA REDUIRE TEXTE PAGE

version 11.2

WA REDUIRE TEXTE PAGE ({*; }objet)

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)

Description

La commande WA REDUIRE TEXTE PAGE réduit la taille du texte affiché dans la zone Web désignée par les paramètres * et objet.

Sous Mac OS, la portée de cette commande est la session 4D : le paramétrage effectué n'est pas conservé après la fermeture de l'application 4D.

Sous Windows, la portée de cette commande est globale : le paramétrage est conservé après la fermeture de l'application 4D.

Référence

WA AGRANDIR TEXTE PAGE.

WA URL precedent disponible

WA URL precedent disponible ({*; }objet) → Booléen

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Booléen	←	Vrai s'il existe un URL précédent dans la séquence d'URLs ouverts, Faux sinon

Description

La commande WA URL precedent disponible permet de savoir s'il existe un URL précédent disponible dans la séquence d'URLs ouverts dans la zone Web désignée par les paramètres * et objet.

La commande retourne Vrai si un URL existe et Faux sinon. Cette commande permet notamment, dans la cadre d'une interface personnalisée, d'activer ou d'inactiver des boutons de navigation.

Référence

WA OUVRIR URL PRECEDENT, WA URL suivant disponible.

WA URL suivant disponible

WA URL suivant disponible ({*; }objet) → Booléen

Paramètre	Type		Description
*	*	→	Si spécifié, objet est un nom d'objet (chaîne) Si omis, objet est une variable
objet	Objet de formulaire	→	Nom d'objet (si * est spécifié) ou Variable (si * est omis)
Résultat	Booléen	←	Vrai s'il existe un URL suivant dans la séquence d'URLs ouverts, Faux sinon

Description

La commande WA URL suivant disponible permet de savoir s'il existe un URL suivant disponible dans la séquence d'URLs ouverts dans la zone Web désignée par les paramètres * et objet.

La commande retourne Vrai si un URL existe et Faux sinon. Cette commande permet notamment, dans la cadre d'une interface personnalisée, d'activer ou d'inactiver des boutons de navigation.

Référence

WA OUVRIR URL SUIVANT, WA URL precedent disponible.

69

Codes d'erreurs

Le tableau suivant liste les codes et les messages des erreurs de syntaxe qui peuvent survenir lors de l'exécution de votre code en mode Développement ou Application. Quelques erreurs peuvent se produire en mode interprété seulement, quelques-unes en mode compilé seulement et les autres dans les deux modes. Ces erreurs peuvent être interceptées par une méthode d'appel sur erreur installée par la commande APPELER SUR ERREUR.

Code	Description
1	Il manque une parenthèse ouvrante.
2	Il manque un champ.
3	Cette fonction ne peut être appliquée que sur un champ appartenant à une sous-table.
4	Les arguments de la liste doivent tous être du même type.
5	Impossible de déterminer sur quelle table appliquer cette fonction
6	Cette fonction ne peut être exécutée que sur un champ de type sous-table.
7	Il manque un argument de type numérique.
8	Il manque un argument de type alphanumérique.
9	Il manque le résultat d'une condition.
10	Cette fonction ne peut être appliquée à ce type de données.
11	Cette fonction ne peut être appliquée entre deux conditions.
12	Cette fonction ne peut être appliquée entre deux arguments numériques.
13	Cette fonction ne peut être appliquée entre deux arguments alphanumériques.
14	Cette fonction ne peut être appliquée entre deux arguments de type date.
15	Les arguments de cette opération ne sont pas compatibles.
16	Ce champ ne possède pas de lien.
17	Il manque une table.
18	Les types sont incompatibles.
19	Le champ n'est pas indexé.
20	Il manque le signe égal (=).
21	Cette méthode n'existe pas.
22	Les champs doivent appartenir à la même table (ou à la même sous-table) pour un tri ou un graphe.
23	Il manque le signe inférieur (<) ou supérieur (>).
24	Il manque un point-virgule (;).
25	Il y a trop de champs pour le tri.
26	Le champ ne doit pas être de type image, texte, BLOB ou sous-table.
27	Le nom du champ doit être préfixé par le nom de la table auquel il appartient.
28	Le champ doit être du type numérique.
29	La valeur doit être égale à 1 ou 0.
30	Il manque une variable.

31 Aucune barre de menus ne porte ce numéro.
32 Il manque une date.
33 Méthode ou fonction non implémentée
34 Les fichiers comptables ne sont pas ouverts.
35 La table et l'ensemble ne sont pas associés.
36 Nom de table incorrect
37 Il manque le signe d'affectation (:=).
38 Ceci est une fonction et non une méthode.
39 Cet ensemble n'existe pas.
40 Ceci est une méthode et non une fonction.
41 Il manque une variable ou un sous-champ.
42 L'enregistrement ne peut pas être dépilé.
43 La fonction est introuvable.
44 La méthode est introuvable.
45 Il manque une variable ou un champ.
46 Il manque un argument de type alphanumérique ou numérique.
47 Le champ doit être de type alphanumérique.
48 Erreur de syntaxe
49 Impossible d'utiliser cet opérateur ici
50 Ces opérateurs ne peuvent pas être utilisés conjointement.
51 Ce module n'est pas implémenté.
52 Il manque un argument de type tableau.
53 L'indice du tableau est en dehors des limites.
54 Les arguments sont incompatibles.
55 Il manque un argument de type booléen.
56 Il manque un champ, une variable ou une table.
57 Il manque un opérateur.
58 Il manque une parenthèse fermante.
59 Type d'argument inattendu
60 Impossible de passer un paramètre ou une variable locale à une commande EXECUTER sur une base compilée
61 Impossible de modifier le type d'un tableau dans une base compilée
62 Impossible d'appliquer cette commande à une sous-table
63 Le champ n'est pas indexé.
64 Il manque un champ ou une variable de type image.
65 La valeur doit comporter 4 caractères.
66 La valeur doit être composée d'au plus 3 caractères.
67 Cette commande ne peut pas être exécutée sur 4D Server.
68 Il manque une liste.
69 Il manque une référence d'une fenêtre externe.
70 Cette fonction ne peut être appliquée entre deux arguments de type image.
71 La commande FIXER TAQUET IMPRESSION peut uniquement être appelée dans l'en-tête d'un formulaire en cours d'impression.

72 Il manque un tableau pointeur.
73 Il manque un tableau numérique.
74 La taille des tableaux ne correspond pas.
75 Pas de pointeur sur les tableaux locaux.
76 Type de tableau erroné.
77 Nom de variable erroné.
78 Paramètre de tri invalide.
79 Cette commande ne peut pas être exécutée pendant le dessin d'une liste.
80 Trop de lignes de recherche.
81 Le formulaire n'a pas été trouvé.

Astuces

Certains codes d'erreurs signalent des erreurs de syntaxe dûes à des fautes de frappe. Par exemple, vous obtenez l'erreur 37 ("Il manque le signe d'affectation (:=).") si vous exécutez l'expression v=0 alors que vous vouliez écrire v:=0. Dans ce cas, vous éliminez l'erreur en corrigeant votre code dans l'éditeur de méthodes.

Certains codes d'erreurs signalent de simples erreurs de programmation. Par exemple, vous obtenez l'erreur 5 ("Impossible de déterminer sur quelle table appliquer cette fonction.") si vous avez exécuté une commande telle que AJOUTER ENREGISTREMENT sans indiquer de nom de table dans le paramètre correspondant, et vous n'avez pas défini de table par défaut à l'aide de la commande TABLE PAR DEFAUT. Dans ce cas, vous corrigez l'erreur en définissant une table par défaut ou en passant un nom de table dans le paramètre correspondant.

Certains codes d'erreurs signalent des erreurs liées à la structure de la base. Par exemple, vous obtenez l'erreur 16 ("Ce champ ne possède pas de lien.") si vous appliquez la commande CHARGER SUR LIEN à un champ qui n'est pas lié à un autre champ. Dans ce cas, vous éliminez l'erreur en modifiant votre code ou en créant un lien à partir du champ.

Certaines erreurs qui surviennent ne stoppent pas toujours l'exécution de votre code au "bon" endroit. Par exemple, si dans une sous-routine vous recevez l'erreur 53 ("L'indice du tableau est en dehors des limites.") sur la ligne vpChamp:=Champ($1;$2), l'erreur est dûe à des numéros incorrects de table ou de champ passés à la sous-routine en tant que paramètres. Donc, l'erreur se trouve dans la méthode appelante et non à l'endroit où l'erreur est détectée. Dans ce cas, tracez votre code dans la fenêtre de débogage et recherchez la ligne qui contient l'erreur, puis corrigez-la dans l'éditeur de méthodes.

Référence

APPELER SUR ERREUR.

Erreurs de la base de données (-10600 -> 4004)

Le tableau suivant liste les codes d'erreurs générées par le moteur de base de données de 4D. Ces erreurs de bas niveau peuvent se produire lors d'opérations liées au moteur telles que des interruptions utilisateur, des erreurs de privilèges ou des objets endommagés.

Code	Description
-10600	Impossible de lire ce BLOB. Il est peut-être endommagé.
-10517	Echec lors de la synchronisation du dossier {folder_name}.
-10516	Le serveur est en cours de maintenance, veuillez vous reconnecter plus tard.
-10515	Votre tentative de connexion à 4D Server a été refusée
-10514	Le nombre maximum d'utilisateurs simultanés pour votre licence a été atteint
-10513	On ne peut appeler la commande {nom_cmd} d'un 4D distant
-10512	L'encodage n'est pas supporté
-10511	La commande "" ne peut pas être appelée depuis un composant
-10510	Impossible de charger le composant ""
-10509	Impossible d'ouvrir la base de données ""
-10508	Méthode projet introuvable
-10507	Cette version n'autorise pas l'ouverture d'une base compilée
-10506	Limite de la version Standard Edition
-10505	Client et serveur ont des numéros de version incompatibles
-10504	Numéro de page d'index non valide
-10503	Numéro d'enregistrement non valide
-10502	Structure d'enregistrement non valide
-10501	Structure d'index non valide.
-10500	Adresse de donnée non valide.
-9999	Disque saturé. Impossible de sauvegarder l'enregistrement.
-9998	La clé d'index existe déjà.
-9997	Le nombre maximum d'enregistrements est atteint.
-9996	La pile est pleine (trop d'appels récursifs ou en cascade).
-9995	Limite de la version de démonstration.
-9994	Communication série interrompue par l'utilisateur. L'utilisateur a appuyé sur les touches Ctrl+Alt+Maj (Windows) ou Commande+Option+Maj (Mac OS).
-9993	Barre de menus endommagée (la base de données doit être réparée).
-9992	Ce mot de passe existe déjà.
-9991	Vous n'avez pas l'autorisation d'accès.
-9990	Dépassement du délai en réception.
-9989	Structure de la base invalide (la base de données doit être réparée).
-9988	Impossible de charger ce formulaire.
-9987	D'autres enregistrements sont liés à celui-ci.

-9986 En attente du déverrouillage d'un enregistrement par le process n°
-9985 Détection d'une boucle lors de la suppression
-9984 Détection d'une clé déjà existante lors d'une transaction.
-9983 Attention ! Vous avez installé deux fois le même package de routines externes.
-9982 Enregistrement non chargé car hors sélection pour le poste client.
-9981 Table de définition de nom/numéro de champ envoyée par le poste client incorrecte.
-9980 Création de table impossible car la structure est verrouillée.
-9979 Impossible d'afficher les informations utilisateur.
-9978 Mot de passe incorrect.
-9977 Cette sélection n'existe pas.
-9976 Cette commande ne peut être exécutée car la base est en cours de sauvegarde.
-9975 Page d'index de transaction non chargeable.
-9974 Cet enregistrement vient d'être détruit.
-9973 Mauvaise ressource TRIC.
-9972 Le numéro de la table est en-dehors de l'intervalle défini par le poste client.
-9971 Le numéro du champ est en-dehors de l'intervalle défini par le poste client.
-9970 Le champ n'est pas indexé.
-9969 Type de champ incorrect.
-9968 Numéro d'enregistrement hors sélection.
-9967 L'enregistrement ne peut pas être modifié car il ne peut pas être chargé.
-9966 Les types sont incompatibles.
-9965 Table de définition de recherche incorrecte envoyée par un poste client.
-9964 Table de définition de tri incorrecte envoyée par un poste client.
-9963 Numéro d'enregistrement non valide.
-9962 Pas de sauvegarde possible car le serveur quitte.
-9961 Le process de sauvegarde n'est pas démarré.
-9960 Aucun plug-in de sauvegarde n'est installé.
-9959 Le process de sauvegarde est déjà démarré.
-9958 Le process ne peut être démarré.
-9957 L'énumération est verrouillée.
-9956 Les versions de 4D Server et 4D Client sont incompatibles.
-9955 QuickTime n'est pas installé.
-9954 Aucun enregistrement courant.
-9953 Il n'y a pas de fichier d'historique.
-9952 Mauvais en-tête du segment principal.
-9951 Ce champ ne possède pas de lien.
-9950 Le numéro d'ordre de ce segment de données n'est pas le bon.
-9949 Erreur de licence ou de privilège.
-9948 Une fenêtre modale est active.
-9947 L'option “Autoriser les connexions 4D Open” n'est pas sélectionnée.
-9946 Impossible d'effacer cette sélection temporaire car elle n'existe pas.

-9945 Erreur 4D Runtime CD-ROM, l'écriture de données est impossible.
-9944 Cet utilisateur n'appartient pas au groupe d'accès par 4D Open.
-9943 Erreur de version de plug-in de connectivité 4D.
-9942 Licence 4D Client incompatible avec cette version de 4D Server.
-9941 Sélecteur EX_GESTALT inconnu.
-9940 L'initialisation de l'extension 4D a échoué.
-9939 Routine externe introuvable.
-9938 L'enregistrement courant a été modifié depuis le trigger.
-9937 Le système de mots de passe est verrouillé par un autre utilisateur.
-9936 Le password extern code ne correspond pas à celui de la base
-9935 Le fichier XML n'est pas valide ou n'est pas bien formé.
-9934 Le fichier XML n'est pas bien formé.
-9933 Le fichier XML n'est pas valide.
-9932 La DLL XML n'est pas chargée.
-9931 L'index pour cet attribut est invalide.
-9930 Il n'existe pas d'attribut de ce nom pour cet élément.
-9929 L'index pour cet élément est invalide.
-9928 Le nom de l'élément est inconnu.
-9927 L'élément référencé n'est pas le "root".
-9926 L'élément référencé est invalide.
-9925 L'élément référencé est nul.
-9924 Le fichier doit être ouvert en lecture seule
-9923 Le nom de l'attribut est invalide
-9922 Manque le paramètre valeur dans la définition des attributs
-9921 Tentative d'écrire un prologue XML dans un document non vide
-9920 Le type du noeud est incorrect
-9919 Cet encodage n'est pas supporté.
-9918 Le nom de l'élément est incorrect.
-9917 Le type du tableau passé en paramètre est incorrect.
-9916 L'élément n'est pas ouvert.
-9915 La référence du document est incorrecte.
-9914 Erreur interne
-9913 Erreur réseau
-9912 Erreur HTTP
-9911 Erreur de l'analyseur xml
-9910 Erreur Web Service
-9909 Pas de fenêtre disponible pour exécuter le formulaire.
-9855 Valeur incorrecte pour le paramètre numéro 5.
-9854 Valeur incorrecte pour le paramètre numéro 4.
-9853 Valeur incorrecte pour le paramètre numéro 3.
-9852 Valeur incorrecte pour le paramètre numéro 2.
-9851 Valeur incorrecte pour le paramètre numéro 1.
-9850 La zone passée à cette commande externe est incorrecte.

-9800 L'un des process a modifié les droits d'accès.
-9759 La bibliothèque d'objets n'a pas pu être ouverte.
-9758 Le formulaire utilisateur existe déjà.
-9757 Le formulaire utilisateur n'existe pas.
-9756 Il n'y a pas de fichier de structure utilisateur.
-9755 Le formulaire utilisateur n'a pas de nom.
-9754 Cette commande ne peut pas être appelée depuis une fenêtre de dialogue.
-9753 Le formulaire source n'existe pas.
-9752 Le formulaire utilisateur ne peut pas être créé.
-9751 Le formulaire source n'est pas accessible pour l'utilisateur.
-9750 Le formulaire source n'est pas modifiable.
-1 Point d'entrée non valide utilisé par un plug-in

1001 Pas de colonne à importer dans la table {TableName}
1002 Table d'import incorrecte
1003 Pas de colonne à exporter dans la table {TableName}
1004 Table d'export incorrecte
1006 Impossible de sauvegarder la structure de la base {BaseName}
1006 Interruption générée par l'utilisateur.
1007 Impossible de créer le fichier de données de la base : {BaseName}
1008 Segment de données incorrect dans la base {BaseName}
1009 La mémoire est pleine
1010 Impossible de charger la définition des tables de la base {BaseName}
1011 >Impossible d'ouvrir le fichier de données de la base {BaseName}
1012 Le segment de données est plein dans la base : {BaseName}
1013 Impossible de sauvegarder le segment de données dans la base {BaseName}
1014 Impossible de lire le segment de données de la base {BaseName}
1015 En-tête de base de données incorrect dans la base {BaseName}
1016 Impossible de créer la table dans la base {BaseName}
1017 Impossible de lire la liste des index de la base {BaseName}
1018 Impossible d'écrire la liste des index de la base {BaseName}
1019 Référence de table incorrecte dans la base {BaseName}
1020 Référence de champ incorrecte dans la base {BaseName}
1021 Type d'index invalide dans la base {BaseName}
1022 Nom de champ invalide pour la table {TableName} de la base {BaseName}
1023 Nom de base de données invalide
1024 Impossible d'ouvrir la structure de la base {BaseName}
1025 Impossible de créer la structure de la base {BaseName}
1026 Impossible de charger la 'bit selection' de la base {BaseName}
1027 Impossible de charger l'ensemble de la base {BaseName}
1028 Impossible de modifier l'ensemble de la base {BaseName}
1029 Impossible de sauvegarder l'ensemble de la base {BaseName}

1030 Impossible de sauvegarder le BLOB {BlobNum} dans la table {TableName} de la base {BaseName}
1031 Impossible de charger le BLOB {BlobNum} dans la table {TableName} de la base {BaseName}
1032 Impossible d'allouer le BLOB {BlobNum} dans la table {TableName} de la base {BaseName}
1033 Impossible de charger la table de bits des données de la base {BaseName}
1034 Impossible de sauvegarder la table de bits des données de la base {BaseName}
1035 Impossible de charger la table de bits des données de la base {BaseName}
1036 Impossible de sauvegarder la table de bits des données dans la base {BaseName}
1037 Impossible de fermer le segment de données de la base {BaseName}
1038 Impossible de supprimer le segment de données de la base {BaseName}
1039 Impossible d'ouvrir le segment de données de la base {BaseName}
1040 Impossible de créer le segment de données de la base {BaseName}
1041 Impossible d'allouer l'espace dans le segment de données
1042 Impossible de libérer l'espace dans le segment de données de la base {BaseName}
1043 Le fichier est protégé en écriture
1044 Impossible d'accéder au champ dans l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1045 Le code de définition du champ est manquant dans l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1046 Impossible de sauvegarder l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1047 Impossible de charger l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1048 Impossible d'allouer l'enregistrement de la table {TableName} de la base {BaseName}
1049 Impossible de mettre à jour l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1050 En-tête incorrect pour l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1051 Impossible de sauvegarder la définition de la table {TableName} de la base {BaseName}
105 Impossible de mettre à jour la définition de la table {TableName} de la base {BaseName}
1053 Le nom du champ existe déjà
1055 Impossible de mettre à jour les valeurs d'index pendant la sauvegarde d'un enregistrement de la table {TableName} de la base {BaseName}
1056 Impossible de mettre à jour les BLOBs pendant la sauvegarde d'un enregistrement de la table {TableName} de la base {BaseName}
1057 Impossible de supprimer les BLOBs pendant la sauvegarde ou la suppression d'un enregistrement de la table {TableName} de la base {BaseName}
1058 Impossible d'ajouter le champ dans la table {TableName} de la base {BaseName}
1059 Impossible d'allouer la table dans la mémoire

1061 Impossible d'allouer l'enregistrement dans la mémoire
1062 Impossible de supprimer complètement la table {TableName} de la base {BaseName}
1063 >Impossible de verrouiller l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1064 Impossible de déverrouiller l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1065 Impossible de supprimer l'enregistrement {RecNum} de la table {TableName} de la base {BaseName}
1066 L'enregistrement {RecNum} est verrouillé dans la table {TableName} de la base {BaseName}
1067 La recherche séquentielle n'a pas pu se terminer dans la table {TableName} de la base {BaseName}
1068 Impossible de sauvegarder l'en-tête de la table {TableName} de la base {BaseName}
1069 Impossible d'importer les données de la table {TableName} de la base {BaseName}
1070 Impossible de charger l'en-tête de l'index de la base {BaseName}
1071 Impossible de sauvegarder l'en-tête de l'index {IndexName} de la base {BaseName}
1072 Impossible de lire l'adresse de la page d'index {IndexName} de la base {BaseName}
1073 Impossible d'écrire l'adresse de la page d'index {IndexName} de la base {BaseName}
1074 Impossible de supprimer l'index {IndexName} de la base {BaseName}
1075 Impossible de trier l'index {IndexName} de la base {BaseName}
1076 Impossible de charger la page d'index {IndexName} de la base {BaseName}
1077 Impossible de sauvegarder la page d'index {IndexName} de la base {BaseName}
1078 Impossible d'insérer la clé dans l'index {IndexName} de la base {BaseName}
1079 Impossible de supprimer la clé de l'index {IndexName} de la base {BaseName}
1080 Impossible de supprimer complètement l'index {IndexName} de la base {BaseName}
1081 Impossible d'accomplir le balayage de l'index {IndexName} de la base {BaseName}
1082 Impossible d'accomplir le tri de l'index {IndexName} de la base {BaseName}
1083 Impossible d'insérer la clé dans la page d'index {IndexName} de la base {BaseName}
1084 Impossible de supprimer la clé dans la page d'index {IndexName} de la base {BaseName}
1085 Impossible de charger le cluster d'index de la base {BaseName}
1086 Impossible d'ajouter au cluster d'index de la base {BaseName}
1087 Impossible de supprimer dans le cluster d'index de la base {BaseName}
1088 L'index {IndexName} est invalide
1089 Erreur de syntaxe SQL (Obsolète)
1090 Mot-clé SQL non trouvé (Obsolète)
1091 Non implémenté
1092 Impossible d'enregistrer le code
1093 Opération en cours annulée par l'utilisateur
1094 Conflit de transaction
1095 Nom de table invalide dans la base {BaseName}
1096 La table {TableName} est verrouillée dans la base {BaseName}
1097 La base {BaseName} est verrouillée

1098 L’adresse de données est invalide dans la base {BaseName}
1099 L’enregistrement est vide
1100 Champ source incorrect
1101 Champ destination incorrect
1102 Nom de lien invalide
1103 Types de champ incompatibles
1104 Le lien est vide
1105 Impossible de charger la liste des liens à partir de la base {BaseName}
1106 Impossible de sauvegarder la liste des liens dans la base {BaseName}
1107 Requête et verrouillage incomplets, un enregistrement au moins était verrouillé
1108 Enregistrement invalide
1109 Numéro d’enregistrement incorrect
1110 Le lien existe déjà dans la base {BaseName}
1111 L’index existe déjà dans la base {BaseName}
1112 Opérateur de comparaison incorrect
1113 Fin du buffer de données
1114 Numéro de version DB4D incorrect
1115 Clé dupliquée
1116 Le champ obligatoire est Null dans l'enregistrement {RecNum} de la table {TableName}
1117 Impossible d’appliquer l’attribut Obligatoire au champ de la table {TableName}
1118 Impossible d’obtenir un accès exclusif à la table {TableName}
1119 Impossible de vérifier l’intégrité référentielle de la table {TableName}
1120 Intégrité référentielle : il y a encore des clés externes correspondant à la clé primaire de l'enregistrement {RecNum} de la table {TableName}
1121 Impossible de supprimer tous les enregistrements de la sélection de la table {TableName}
1122 Le BLOB est Null
1123 Contexte de base de données incorrect
1124 Référence de lien invalide
1125 Nom d’enregistrement incorrect dans la table {TableName}
1126 Type de champ incorrect
1127 Impossible de charger les propriétés additionnelles
1128 Impossible de sauvegarder les propriétés additionnelles
1129 Le numéro du sous-enregistrement est en dehors des limites
1130 Nom dupliqué pour l'index {IndexName} dans la base {BaseName}
1131 Nom invalide pour l'index {IndexName} dans la base {BaseName}
1132 Valeur de clé invalide pour l'index {IndexName}
1133 Type incorrect pour l'index {IndexName}
1134 Accesseur invalide
1135 L’accesseur est en lecture seulement
1136 Valeur Null non acceptée
1137 THIS est Null

1138 La sélection est Null
1139 La base de données {BaseName} est protégée en écriture
1140 La base de données {BaseName} est en cours de fermeture
1141 Transaction invalide
1142 La limite de tableau est dépassée
1143 Le créateur des valeurs du tableau est manquant
1144 Impossible de construire la sélection de la table {TableName}
1145 Objet actuellement indisponible
1146 Le fichier de données ne correspond pas au fichier de structure
1147 Impossible de lancer le serveur d'écoute réseau
1148 Impossible de lancer le serveur
1149 Pas de serveur d'écoute réseau
1150 Le process est en train de mourir
1151 Balise de requête invalide
1152 Numéro de contexte invalide
1153 Espace disque temporaire insuffisant
1154 L'ensemble de données est Null
1155 Aucune clé primaire ne correspond à la clé externe
1156 Le type du champ {FieldName} de la table {TableName} n'accepte pas l'attribut Unique
1157 Le type du champ {FieldName} de la table {TableName} n'accepte pas l'attribut NEVER NULL (jamais Null)
1158 Impossible de modifier la définition de la clé primaire de la table {TableName}
1159 Le nombre maximal d'enregistrements a été atteint pour la table {TableName}
1160 Le nombre maximal de BLOBs a été atteint pour la table {TableName}
1161 L'indice est en dehors des limites
1162 La requête est invalide
1163 L'enregistrement est Null
1164 L'objet est Null
1165 Propriétaire incorrect pour cet objet
1166 L'objet n'était pas verrouillé
1167 L'objet est verrouillé par un autre contexte
1168 Erreur interne sur une connexion distante
1169 Numéro de table invalide
1170 Numéro de champ invalide
1171 Numéro de base de données invalide
1172 Paramètre invalide
1173 L'écriture du cache est incomplète
1174 L'écriture des données est incomplète
1175 L'écriture de la structure est incomplète
1176 Le fichier d'historique est invalide pour la base {BaseName}
1177 Le fichier d'historique est introuvable pour la base {BaseName}

1178 La dernière opération du fichier d'historique ne correspond pas à la base {BaseName}
1179 Le fichier d'historique ne correspond pas à la base {BaseName}
1180 Les données de la table ont été supprimées
1181 Les clés ne sont pas uniques dans l'index {IndexName}
1182 Impossible de créer un fichier d'historique pour la base {DataBase}
1183 Impossible d'écrire dans le fichier d'historique de la base {DataBase}
1184 Impossible de déposer la table {TableName} dans la base {DataBase}
1185 La base de données distante ne peut pas être ouverte
1186 Le fichier d'historique ne peut pas être intégré à la base {DataBase}
1187 L'opération interne sur l'ensemble est incomplète
1188 Le tableau ne peut pas être sauvegardé
1189 Le tableau ne peut pas être chargé
1190 L'en-tête du numéro de séquence ne peut pas être chargé
1191 Impossible de sélectionner l'enregistrement
1192 L'enregistrement ne peut pas être créé
1193 Impossible d'accomplir sélection vers tableau de la table {TableName} de la base {BaseName}
1194 Impossible d'accomplir tableau vers sélection de la table {TableName} de la base {BaseName}
1195 Impossible d'accomplir le tri séquentiel
1196 La sélection ne peut pas être verrouillée
1197 La clé d'index ne peut pas être chargée
1198 La clé d'index ne peut pas être sauvegardée
1199 La clé d'index ne peut pas être construite
1200 La requête ne peut pas se terminer
1201 La requête ne peut pas être analysée
1202 La formule n'a pas pu être effectuée sur cette colonne
1203 La requête n’a pas pu se terminer
1204 La requête n’a pas pu être analysée
1205 Impossible d'obtenir toutes les valeurs distinctes de la table {TableName} de la base {BaseName}
1206 Impossible de construire le tableau de valeurs de la table {TableName} de la base {BaseName}
1207 La sélection ne peut pas être chargée
1208 Impossible d'envoyer les données
1209 Impossible de recevoir des données
1210 Impossible d'envoyer une requête
1211 Impossible de créer une connexion
1212 L'index {IndexName} ne peut pas être créé rapidement dans la base {BaseName}
1213 Impossible de construire les clés d'index distinctes
1214 La sélection ne peut pas être triée dans la table {TableName} de la base {BaseName}
1215 La table d'adresse ne peut pas être chargée

1216 La table d'adresse ne peut pas être modifiée
1217 mpossible d'allouer une nouvelle entrée à la table d'adresse
1218 Impossible d'allouer une entrée vide de la table d'adresse
1219 L'enregistrement temporaire de transaction ne peut pas être sauvegardé
1220 Le blob temporaire de transaction ne peut pas être sauvegardé
1221 L'enregistrement temporaire de transaction ne peut pas être chargé
1222 Le blob temporaire de transaction ne peut pas être chargé
1223 La transaction ne peut pas être démarrée
1224 La transaction ne peut pas être validée
1225 Impossible de lire la propriété additionnelle
1226 Impossible d'écrire la propriété additionnelle
1227 Le nom de la table est déjà utilisé
1228 Impossible de lire la liste des clés NULL de l'index {IndexName}
1229 Impossible de modifier la liste des clés NULL de l'index {IndexName}
1230 Clé invalide pour l'index {IndexName}
1231 Impossible d'écrire le fichier d'historique
1232 Le contexte est NULL
1233 La base {BaseName} ne peut pas être verrouillée
1234 Référence de champ incorrecte dans l'enregistrement# {RecNum} de la table : {TableName} de la base : {BaseName}
1235 Référence de champ incorrecte dans la table : {TableName} de la base : {BaseName}
1236 Impossible de lire les données du fichier de transaction temporaire
1237 Le produit cartésien échoué
1238 Impossible de fusionner sélections
1239 Le format de la base {BaseName} ne peut pas être mis à jour en mode lecture seulement
1240 En-tête incorrect
1241 CheckSum incorrect
1243 Impossible de charger la table des données de la base : {BaseName}
1244 La liste de contraintes des clés étrangères n'est pas vide pour la table : {TableName}dans la base : {BaseName}
1245 L'entrée d'adresse n'est pas vide
1246 Impossible de pré-allouer l'adresse
1247 Impossible de mettre à jour le nouvel enregistrement dans la table {TableName} de la base {BaseName}
1248 Impossible de sauvegarder le nouvel enregistrement dans la table {TableName} de la base {BaseName}
1249 Impossible de sauvegarder le sous-enregistrement
1250 Impossible de sauvegarder l'enregistrement
1251 Impossible de verrouiller la définition de l'objet de structure
1252 Impossible de déverrouiller la définition de l'objet de structure
1253 Numéro de lien invalide

1254 Référence circulaire de la table d'adresse des enregistrements de la table {TableName} de la base {BaseName}
1255 Référence circulaire dans la table d'adresse des blobs de la table {TableName} de la base {BaseName}
1256 Nom de schéma dupliqué dans la base {BaseName}
1257 Le schéma ne peut être sauvegardé dans la base {BaseName}
1258 Le schéma ne peut être supprimé dans la base {BaseName}
1259 Le schéma ne peut être renommé dans la base {BaseName}
1260 Le fichier d'historique est trop récent pour la base {BaseName}
1261 Le fichier d'historique est trop ancien pour la base {BaseName}

1300 Type de sélection invalide
1301 Le tableau est trop grand
1302 La taille du tableau ne correspond pas
1303 ID de la sélection invalide
1304 Partie de la sélection invalide

4001 Numéro de table non valide utilisé par un plug-in
4002 Numéro d'enregistrement non valide utilisé par un plug-in
4003 Numéro de champ invalide utilisé par un plug-in
4004 Un plug-in a requis l'enregistrement courant d'une table alors qu'il n'y en a pas

Notes

(1) Bien que certaines de ces erreurs signalent des problèmes sérieux — par exemple (-10502), Structure d'enregistrement non valide — la plupart sont relativement courantes et peuvent peuvent être traitées par une méthode projet APPELER SUR ERREUR. Par exemple, vous intercepterez fréquemment l'erreur –9998, La clé d'index existe déjà si votre application laisse la possibilité de créer des valeurs identiques pour une table qui contient un champ indexé ayant la propriété Unique.

(2) Certaines de ces erreurs ne se produisent jamais au niveau du langage de 4D. Elles ne surviennent et ne peuvent être traitées qu'à un bas niveau par des routines du moteur de la base ou pendant l'utilisation, par exemple, de 4D Open.

(3) L'erreur -10503, Numéro d'enregistrement non valide signifie généralement que votre code (par exemple la commande ALLER A ENREGISTREMENT) tente d'accéder à un enregistrement qui n'existe pas ou plus. Dans certains cas, plus rares, cette erreur peut signifier que la base doit être réparée.

(4) L'erreur -9999 Disque saturé. Impossible de sauvegarder l'enregistrement se produit lorsque le fichier de données de votre base est plein ou placé sur un volume plein. Cette erreur peut également être générée si le fichier de données est verrouillé ou stocké sur un volume verrouillé.

Référence

APPELER SUR ERREUR.

Le moteur SQL de 4D retourne des erreurs spécifiques, listées ci-dessous. Ces erreurs peuvent être interceptées à l'aide d'une méthode gestion d'erreurs installée par la commande APPELER SUR ERREUR et analysées via la commande LIRE PILE DERNIERE ERREUR.

Erreur génériques

1001 INVALID ARGUMENT
1002 INVALID INTERNAL STATE
1003 NOT RUNNING
1004 Accès refusé
1005 FAILED TO LOCK SYNCHRONIZATION PRIMITIVE
1006 FAILED TO UNLOCK SYNCHRONIZATION PRIMITIVE
1007 SQL SERVER IS NOT AVAILABLE
1008 COMPONENT BRIDGE IS NOT AVAILABLE
1009 REMOTE SQL SERVER IS NOT AVAILABLE
1010 L'exécution a été interrompue par l'utilisateur.

Erreurs sémantiques

1101 La table '{key1}' n'existe pas dans la base de données.
1102 La colonne '{key1}' n'existe pas.
1103 La table '{key1}' n'est pas déclarée dans la clause FROM.
1104 La référence vers le nom de la colonne '{key1}' est ambigüe.
1105 L'alias '{key1}' de la table est identique au nom de la table.
1106 Alias de tables dupliqués
1107 Table dupliquée dans la clause FROM - '{key1}'.
1108 L'opération {key1} {key2} {key3} n'est pas de type valide.
1109 Index invalide dans la clause ORDER BY - {key1}.
1110 La fonction {key1} attend un paramètre, pas {key2}.
1111 Le paramètre {key1} de type {key2} dans l'appel de la fonction {key3} n'est pas convertible implicitement en {key4}.
1112 Fonction inconnue - {key1}.
1113 Division par zéro.
1114 Le tri par article indexé dans la liste SELECT n'est pas autorisé - article {key1} dans la clause ORDER BY.
1115 DISTINCT NOT ALLOWED
1116 Les fonctions statistiques imbriquées ne sont pas autorisées dans la fonction statistique {key1}.
1117 La sous requête ne doit pas contenir de fonctions de colonnes dans le contexte de sa super-requête.
1118 Impossible de mixer des opérations de colonne et scalaires.
1119 Index invalide dans la clause GROUP BY - {key1}.
1120 L'index GROUP BY n'est pas autorisé.
1121 GROUP BY n'est pas autorisé avec 'SELECT * FROM ...'.

1122 HAVING n'est pas une expression statistique
1123 La colonne '{key1}' n'est pas une colonne groupée et ne peut pas être utilisée dans la clause ORDER BY.
1124 Impossible de mixer les types {key1} et {key2} dans le prédicat IN.
1125 La séquence d'échappement '{key1}' dans le prédicat LIKE est trop longue. Ce doit être exactement un caractère.
1126 Caractère d'échappement erroné - '{key1}'.
1127 Séquence d'échappement inconnue - '{key1}'.
1128 Les références des colonnes de plus d'une requête dans la fonction statistique {key1} n'est pas autorisé.
1129 L'élément scalaire dans la liste SELECT n'est pas autorisé quand la clause GROUP BY est présente.
1130 Les sous requêtes produisent plus d'une colonne.
1131 La sous requête doit retourner une ligne au maximum mais ici elle renvoie {key1}.
1132 Le nombre de valeurs dans INSERT {key1} ne correspond pas au nombre de colonnes {key2}.
1133 Référence de colonne dupliquée dans la liste INSERT - '{key1}'.
1134 La colonne '{key1}' n'autorise pas les valeurs NULL.
1135 Référence de colonne dupliquée dans la liste UPDATE - '{key1}'.
1136 Table '{key1}' déjà existante.
1137 Colonne '{key1}' dupliquée dans la commande CREATE TABLE.
1138 DUPLICATE COLUMN IN COLUMN LIST
1139 Une seule clé primaire est autorisée.
1140 Nom de clé externe ambigu - '{key1}'.
1141 Le nombre de colonnes {key1} dans la table enfant ne correspond pas au nombre de colonnes {key2} dans la table parent de la clé externe.
1142 Incompatibilité de types de colonne dans la définition de la clé externe. Impossible de lier {key1} dans la table enfant à {key2} dans la table parent.
1143 Impossible de trouver la colonne correspondante dans la table parent pour la colonne '{key1}' dans la table enfant.
1144 Les contraintes UPDATE et DELETE doivent être les mêmes.
1145 FOREIGN KEY DOES NOT EXIST
1146 Valeur invalide pour LIMIT dans la commande SELECT - {key1}.
1147 Valeur invalide pour OFFSET dans la commande SELECT - {key1}.
1148 La clé primaire n'existe pas dans la table '{key1}'.
1149 FAILED TO CREATE FOREIGN KEY
1150 La colonne '{key1}' ne fait pas partie de la clé primaire.
1151 FIELD IS NOT UPDATEABLE
1153 Longueur de type de données incorrecte '{key1}'.
1156 L'auto-increment n'est pas autorisée pour la colonne '{key1}' de type {key2}.
1159 Impossible de supprimer le schéma système.
1160 CHARACTER ENCODING NOT ALLOWED.

Erreurs d'mplémentations

1203 FUNCTIONALITY IS NOT IMPLEMENTED
1204 Echec de la création de l'enregistrement {key1}.
1205 Echec de la mise à jour du champ '{key1}'.

1206 Echec de la suppression de l'enregistrement '{key1}'.
1207 NO MORE JOIN SEEDS POSSIBLE
1208 FAILED TO CREATE TABLE
1209 FAILED TO DROP TABLE
1210 CANT BUILD BTREE FOR ZERO RECORDS
1211 COMMAND COUNT GREATER THAN ALLOWED
1212 FAILED TO CREATE DATABASE
1213 FAILED TO DROP COLUMN
1214 VALUE IS OUT OF BOUNDS
1215 FAILED TO STOP SQL_SERVER
1216 FAILED TO LOCALIZE
1217 Impossible de verrouiller la table pour la lecture.
1218 FAILED TO LOCK TABLE FOR WRITING
1219 TABLE STRUCTURE STAMP CHANGED
1220 FAILED TO LOAD RECORD
1221 FAILED TO LOCK RECORD FOR WRITING
1222 FAILED TO PUT SQL LOCK ON A TABLE
1223 FAILED TO RETAIN COOPERATIVE TASK
1224 FAILED TO LOAD INFILE

Erreurs d'analyse

1301 PARSING FAILED

Erreurs d'accès au langage runtime

1401 COMMAND NOT SPECIFIED
1402 ALREADY LOGGED IN
1403 SESSION DOES NOT EXIST
1404 UNKNOWN BIND ENTITY
1405 INCOMPATIBLE BIND ENTITIES
1406 REQUEST RESULT NOT AVAILABLE
1407 BINDING LOAD FAILED
1408 COULD NOT RECOVER FROM PREVIOUS ERRORS
1409 NO OPEN STATEMENT
1410 RESULT EOF
1411 BOUND VALUE IS NULL
1412 STATEMENT ALREADY OPENED
1413 FAILED TO GET PARAMETER VALUE
1414 INCOMPATIBLE PARAMETER ENTITIES
1415 Le paramètre de la requête n'est pas autorisé ou est manquant.
1416 COLUMN REFERENCE PARAMETERS FROM DIFFERENT TABLES
1417 EMPTY STATEMENT
1418 FAILED TO UPDATE VARIABLE
1419 FAILED TO GET TABLE REFERENCE
1420 FAILED TO GET TABLE CONTEXT
1421 COLUMNS NOT ALLOWED
1422 INVALID COMMAND COUNT
1423 INTO CLAUSE NOT ALLOWED

1424 EXECUTE IMMEDIATE NOT ALLOWED
1425 ARRAY NOT ALLOWED IN EXECUTE IMMEDIATE
1426 COLUMN NOT ALLOWED IN EXECUTE IMMEDIATE
1427 NESTED BEGIN END SQL NOT ALLOWED
1428 RESULT IS NOT A SELECTION
1429 INTO ITEM IS NOT A VARIABLE
1430 VARIABLE WAS NOT FOUND
1431 PTR OF PTR NOT ALLOWED
1432 POINTER OF UNKNOWN TYPE

Erreurs d'analyse de date

1501 SEPARATOR_EXPECTED
1502 FAILED TO PARSE DAY OF MONTH
1503 FAILED TO PARSE MONTH
1504 FAILED TO PARSE YEAR
1505 FAILED TO PARSE HOUR
1506 FAILED TO PARSE MINUTE
1507 FAILED TO PARSE SECOND
1508 FAILED TO PARSE MILLISECOND
1509 INVALID AM PM USAGE
1510 FAILED TO PARSE TIME ZONE
1511 UNEXPECTED CHARACTER
1512 Echec de l'analyse du timestamp.
1513 Echec de l'analyse de la durée.
1551 FAILED TO PARSE DATE FORMAT

Erreurs lexer

1601 NULL INPUT STRING
1602 NON TERMINATED STRING
1603 NON TERMINATED COMMENT
1604 INVALID NUMBER
1605 UNKNOWN START OF TOKEN
1606 NON TERMINATED NAME
1607 NO VALID TOKENS

Erreurs de validation - Erreurs de statut suivant les erreurs directes

1701 Echec de la validation de la table '{key1}'.
1702 Echec de la validation de la clause FROM.
1703 Echec de la validation de la clause GROUP BY.
1704 Echec de la validation de la liste SELECT.
1705 Echec de la validation de la clause WHERE.
1706 Echec de la validation de la clause ORDER BY.
1707 Echec de la validation de la clause HAVING.
1708 Echec de la validation du prédicat COMPARISON.
1709 Echec de la validation du prédicat BETWEEN.
1710 Echec de la validation du prédicat IN.
1712 Echec de la validation du prédicat ALL ANY.

1713 Echec de la validation du prédicat EXISTS.
1714 Echec de la validation du prédicat NULL.
1715 Echec de la validation de la sous-requête.
1716 Echec de la validation de l'article SELECT {key1}.
1717 Echec de la validation de la colonne '{key1}'.
1718 Echec de la validation de la fonction '{key1}'.
1719 Echec de la validation de l'expression CASE.
1720 Echec de la validation du paramètre de la commande.
1721 Echec de la validation du paramètre '{key1}' de la fonction.
1722 Echec de la validation de l'article '{key1}' de la liste INSERT.
1723 Echec de la validation de l'article '{key1}' de la liste UPDATE.
1724 Echec de la validation de la liste des colonnes.
1725 Echec de la validation de la clé étrangère.
1726 Echec de la validation de la commande SELECT.
1727 Echec de la validation de la commande INSERT.
1728 Echec de la validation de la commande DELETE.
1729 Echec de la validation de la commande UPDATE.
1730 Echec de la validation de la commande CREATE TABLE.
1731 Echec de la validation de la commande DROP TABLE.
1732 Echec de la validation de la commande ALTER TABLE.
1733 Echec de la validation de la commande CREATE INDEX.
1734 Echec de la validation de la commande LOCK TABLE.
1735 Echec du calcul du modèle du prédicat LIKE.

Erreurs d'exécution - Erreurs de statut suivant les erreurs directes

1801 Echec de l'exécution de la commande SELECT.
1802 Echec de l'exécution de la commande INSERT.
1803 Echec de l'exécution de la commande DELETE.
1804 Echec de l'exécution de la commande UPDATE.
1805 Echec de l'exécution de la commande CREATE TABLE.
1806 Echec de l'exécution de la commande DROP TABLE.
1807 Echec de l'exécution de la commande CREATE DATABASE.
1808 Echec de l'exécution de la commande ALTER TABLE.
1809 Echec de l'exécution de la commande CREATE INDEX.
1810 Echec de l'exécution de la commande DROP INDEX.
1811 Echec de l'exécution de la commande LOCK TABLE.
1812 Echec de l'exécution de la commande TRANSACTION.
1813 Echec de l'exécution de la clause WHERE.
1814 Echec de l'exécution de la clause GROUP BY.
1815 Echec de l'exécution de la clause HAVING.
1816 Echec de l'agrégation.
1817 Echec de l'exécution de DISTINCT.
1818 Echec de l'exécution de la clause ORDER BY.
1819 Echec de la construction de la requête DB4D.
1820 Echec de calcul du prédicat de la comparaison.
1821 Echec de l'exécution de la sous requête.
1822 Echec du calcul du prédicat de BETWEEN.
1823 Echec du calcul du prédicat de IN.

1824 Echec du calcul du prédicat de ALL/ANY.
1825 Echec du calcul du prédicat de LIKE.
1826 Echec du calcul du prédicat de EXISTS.
1827 Echec du calcul du prédicat de IS NULL.
1828 Echec de l'opération arithmétique.
1829 Echec du calcul l'appel de fonction '{key1}'.
1830 Echec du calcul de l'expression Au cas ou.
1831 Echec du calcul du paramètre de la fonction '{key1}'.
1832 Echec du calcul de l'appel de la fonction 4D.
1833 Echec du tri pendant l'exécution de la clause ORDER BY.
1834 Echec du calcul du hash de l'enregistrement.
1835 Echec de la comparaison des enregistrements.
1836 Echec du calcul de la valeur {key1} de INSERT.
1837 SQL DB4D QUERY FAILED
1838 FAILED TO EXECUTE ALTER SCHEMA COMMAND
1839 FAILED TO EXECUTE GRANT COMMAND

Erreurs de cache

2000 CACHEABLE NOT INITIALIZED
2001 VALUE ALREADY CACHED
2002 CACHED VALUE NOT FOUND
2003 CACHE IS FULL
2004 CACHING IS NOT POSSIBLE

Erreurs de protocole

3000 HEADER NOT FOUND
3001 UNKNOWN COMMAND
3002 ALREADY LOGGED IN
3003 NOT LOGGED IN
3004 UNKNOWN OUTPUT MODE
3005 INVALID STATEMENT ID
3006 UNKNOWN DATA TYPE
3007 STILL LOGGED IN
3008 SOCKET READ ERROR
3009 SOCKET WRITE ERROR
3010 BASE64 DECODING ERROR
3011 SESSION TIMEOUT
3012 FETCH TIMESTAMP ALREADY EXISTS
3013 BASE64 ENCODING ERROR
3014 INVALID HEADER TERMINATOR
3015 INVALID SESSION TICKET
3016 HEADER TOO LONG
3017 INVALID AGENT SIGNATURE
3018 UNEXPECTED HEADER VALUE

Erreurs réseau (-10051 -> -10001)

Le tableau suivant liste les erreurs qui peuvent se produire dans le cadre des connexions réseau.

Code	Description
-10051	Valeur incorrecte dans Get/SetOption
-10050	Option inconnue dans Get/SetOption
-10033	Taille des données incorrecte pendant la lecture
-10031	Désynchronisation pendant la lecture
-10030	Désynchronisation pendant l'écriture
-10021	Aucun serveur trouvé par OP Find 4D server
-10020	Aucun serveur sélectionné par OP Select 4D server
-10003	Paramètres de connexion incorrects
-10002	Interruption de la connexion pour ce process
-10001	Interruption de la connexion à la base en cours

Le tableau suivant liste les codes d'erreurs spécifiques générés par le module de sauvegarde et de restitution de 4D.
Ces erreurs peuvent être interceptées par une méthode installée via la commande APPELER SUR ERREUR.

Code	Description
1401	Le nombre maximal de tentatives de sauvegarde est atteint, la sauvegarde automatique est temporairement désactivée.
1403	Cette base travaille sans fichier d'historique.
1404	Une transaction est ouverte dans ce process. La sauvegarde ne peut pas démarrer.
1405	Le délai d'attente maximum de fin de transaction dans un process concurrent est atteint. L'opération ne peut être exécutée.
1406	La sauvegarde a été annulée par l'utilisateur.
1407	Le dossier de destination est invalide.
1408	Erreur pendant la sauvegarde du fichier d'historique.
1409	Erreur pendant la sauvegarde.
1410	Le fichier de sauvegarde est introuvable, impossible de le vérifier.
1411	Erreur pendant la vérification du fichier de sauvegarde.
1412	Le fichier de sauvegarde d'historique est introuvable, impossible de le vérifier.
1413	Erreur pendant la vérification du fichier de sauvegarde d'historique.
1414	Cette commande ne peut être exécutée que sur 4D Server.
1415	Impossible de sauvegarder l'historique, une opération critique est en cours.
1416	Ce fichier d'historique ne correspond pas à la base ouverte.
1417	Une opération d'intégration du fichier d'historique est en cours. La sauvegarde ne peut pas démarrer.
1420	Impossible d'intégrer l'historique car un enregistrement au moins est verrouillé dans la base.
1421	Cette commande ne peut pas être utilisée en environnement client/serveur.

• Les erreurs 1408 et 1409 proviennent généralement d'une erreur de lecture des fichiers à sauvegarder ou d'une erreur d'écriture avec les fichiers en cours de sauvegarde.
• Les erreurs 1411 et 1413 se produisent lors de la vérification des archives.
Lorsque ces erreurs surviennent, il peut être judicieux de vérifier dans un premier temps la place restante sur le disque et les privilèges d'accès en lecture écriture.

Référence

APPELER SUR ERREUR.

Erreurs du gestionnaire de fichiers du système (-124 -> -33)

Le tableau suivant liste les codes d'erreurs retournés par le gestionnaire de fichiers du système d'exploitation. Ces erreurs peuvent se produire en particulier lorsque vous utilisez les commandes du thème Documents système. Pour plus d'informations, reportez-vous à la section Présentation des documents système.

Code	Description
-124	Tentative d'accès à un volume partagé déconnecté
-121	Un chemin d'accès ne peut être créé
-120	Tentative d'accès à un fichier avec un chemin d'accès spécifiant un répertoire inexistant
-84	Problème physique avec le disque (installation ou formatage incorrect...)
-64	Problème physique avec le disque (installation ou formatage incorrect...)
-61	L'accès en lecture/écriture ne permet pas d'écrire.
-60	Mauvais secteur de répertoire principal. Votre disque est endommagé.
-58	Erreur de fichier système
-57	Tentative d'écriture sur un disque non-Macintosh
-54	Tentative d'écriture dans un fichier verrouillé
-53	Le volume a été éjecté.
-52	Erreur interne du gestionnaire de fichiers (le marqueur de fichiers est perdu).
-51	Tentative d'accès à un fichier avec un numéro de référence de fichier invalide
-49	Fichier déjà ouvert en Lecture/Ecriture.
-48	Tentative d'utilisation du nom d'un fichier déjà supprimé pour renommer un fichier.
-47	Tentative d'accès à un fichier supprimé.
-46	Volume verrouillé par logiciel.
-45	Fichier verrouillé.
-44	Disque physiquement verrouillé.
-43	Fichier non trouvé.
-42	Trop de fichiers ouverts.
-41	Mémoire insuffisante pour ouvrir un nouveau fichier.
-40	Tentative de lecture ou d'écriture avant le début du fichier.
-39	Tentative de lecture après la fin de fichier.
-38	Tentative de lecture ou d'écriture dans un fichier non ouvert.
-37	Nom de fichier ou de volume incorrect. Il est trop long et/ou comporte un caractère
-36	Erreur d'Entrée/Sortie. Il y a probablement un secteur défectueux sur le disque.
-35	Le volume n'existe pas.
-34	Le disque est plein. Il n'y a plus de place disponible sur le disque.

-33 Le répertoire du disque est plein. Vous ne pouvez pas créer de fichier sur le disque. invalide.

Référence

APPELER SUR ERREUR.

Erreurs du gestionnaire de mémoire du système (-117 -> -108)

Le tableau ci-dessous liste les principaux codes d'erreurs retournés par le gestionnaire de mémoire du système d'exploitation.

Code	Description
-117	Problème interne de mémoire. La mémoire est probablement endommagée. Quittez l'application dès que possible. Redémarrez la machine et réouvrez la base.
-111	Problème interne de mémoire. La mémoire est probablement endommagée. Quittez l'application dès que possible. Redémarrez la machine et réouvrez la base. (*)
-109	Problème interne de mémoire. La mémoire est probablement endommagée. Quittez l'application dès que possible. Redémarrez la machine et réouvrez la base.
-108	Mémoire disponible insuffisante. Allouez davantage de mémoire à votre application 4D.

Conseil : Lorsque vous travaillez avec des gros tableaux, des BLOBs, des images ou des ensembles (c'est-à-dire des objets pouvant manipuler de grandes quantités de données), utilisez une méthode projet installée par APPELER SUR ERREUR pour tester l'erreur -108.

(*) Une erreur -111 peut également se produire lorsque vous tentez de lire une valeur dans un BLOB à un offset hors intervalle. Dans ce cas, cette erreur est mineure et vous n'êtes pas obligé de fermer la session de travail. Il vous suffit de corriger l'offset que vous avez passé à la commande BLOB.

Référence

APPELER SUR ERREUR.

Erreurs du gestionnaire d'impression du système (-8192 -> -1)

Le tableau ci-dessous liste les codes d'erreurs retournés par le gestionnaire d'impression du système d'exploitation. Ces codes peuvent être retournés durant l'impression.

Code	Description
-8192	Time out de l'imprimante
-8151	L'imprimante a été initialisée avec une version du driver différente de la vôtre
-8150	Aucune imprimante n'a été sélectionnée
-4101	Imprimante éteinte ou non connectée
-4100	La connexion avec l'imprimante a été interrompue
-193	Fichier de ressources introuvable
-128	Impression interrompue par l'utilisateur
-27	Problème de communication avec l'imprimante
-1	Problème lors de l'enregistrement d'un fichier à imprimer

Référence

APPELER SUR ERREUR.

Erreurs du gestionnaire de ressources du système (-196 -> -1)

Le tableau ci-dessous liste les codes d'erreurs retournés par le gestionnaire de ressources du système d'exploitation.

Code	Description
-196	La ressource ne peut être supprimée
-194	La ressource ne peut être ajoutée
-193	La ressource est endommagée (le fichier doit être réparé)
-192	Ressource introuvable
-1	Impossible d'ouvrir ce fichier de ressources

Référence

APPELER SUR ERREUR.

Erreurs SANE NaN (1 -> 255)

Le tableau ci-dessous liste les codes NaN retournés par le système d'exploitation. Le sigle NaN signifie “Not a Number”. C'est une représentation du Standard Apple Numeric Environment (SANE) qui est appelée lorsqu'une opération produit un résultat se trouvant dans le domaine de SANE.

Code	Description
1	Racine carrée invalide
2	Addition invalide
4	Division invalide
8	Multiplication invalide
9	Reste invalide
17	Conversion d'une chaîne ASCII invalide
20	Conversion d'un nombre de type Comp en virgule flottante
21	Création d'un NaN avec un code zéro
33	Argument invalide passé à une fonction trig
34	Argument invalide passé à une fonction trig inversée
36	Argument invalide passé à une fonction log
37	Argument invalide passé à une fonction xi ou xy
38	Argument invalide passé à une fonction financière
255	Stockage non initialisé

Erreurs du gestionnaire de son du système (-209 -> -203)

Le tableau ci-dessous liste les codes retournés par le gestionnaire de son du système d'exploitation.

Code	Description
-209	Le canal de son est indisponible
-207	Mémoire insuffisante pour jouer le son
-206	Format de ressource son incorrect
-205	Le canal de son est endommagé
-204	La ressource son ne peut être chargée
-203	Trop de commandes de sons

Référence

APPELER SUR ERREUR.

Le tableau ci-dessous fournit le code d'erreur retourné par le gestionnaire de port série du système d'exploitation.

Code	Description
-28	Il n'y a pas de port série ouvert

Référence

APPELER SUR ERREUR.

Le tableau ci-dessous liste certaines erreurs courantes retournées par Mac OS. Lorsqu'une de ces erreurs se produit, il n'est généralement pas possible de poursuivre la session sans redémarrer.

Code	Description
4	Division par zéro
15	Erreur du chargeur de segments : 4D n'a pas pu charger un de ses propres segments de code. Vous devez allouer davantage de mémoire à 4D.
17 à 24	Un élément système est manquant. Vérifiez que votre dossier système a été correctement installé.
25	Mémoire saturée. Vous devez allouer davantage de mémoire à 4D.
28	La pile a été placée dans la heap de l'application. Vous devez allouer davantage de mémoire à 4D.

70

Codes de caractères

Dans les bases de données créées avec 4D v11, le langage ainsi que le moteur de la base de données stockent et manipulent nativement les caractères en Unicode.
Ce principe facilite l'internationalisation des applications 4D. L'Unicode est un jeu de caractères standard unifié qui gère pratiquement toutes les langues usuelles de la planète. Un jeu de caractères est une table de correspondance caractère/valeur numérique, par exemple "a"->1, "b"->2, "5"->15, "oe"->662, etc. Alors qu'en ASCII la valeur numérique de base est typiquement comprise entre 1 et 127, en Unicode la borne haute va au-delà de 65000, ce qui permet de représenter quasiment tous les caractères de toutes les langues.
Il existe différentes manières de coder les valeurs numériques Unicode : UTF-16 les code sur des entiers de 16-bits, UTF-32 sur des entiers de 32-bits et UTF-8 sur des entiers de 8-bits. 4D utilise principalement UTF-16 (comme Windows et Mac OS). Parfois, essentiellement pour des besoins liés au Web, 4D utilise UTF-8 qui a l'avantage de la compacité et de la lisibilité pour les caractères usuels (a-z,0-9).

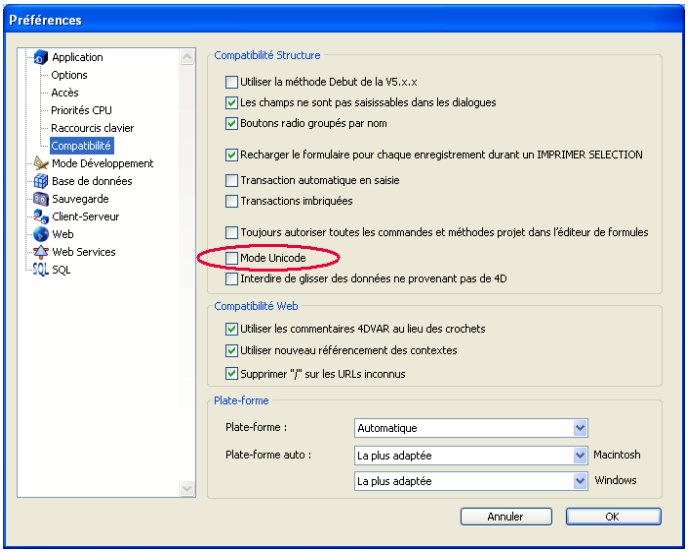
Pour plus d'informations sur la norme Unicode, reportez-vous par exemple à la page suivante :
http://fr.wikipedia.org/wiki/Unicode
Une liste de codes Unicode (page en anglais) :
http://en.wikipedia.org/wiki/List_of_Unicode_characters

Attention : En unicode dans 4D v11, les codes de caractères suivants sont réservés et ne doivent jamais être inclus dans un texte:

0
65534 (FFFE)
65535 (FFFF)

Note de compatibilité : Les bases de données créées avec une version de 4D antérieure à la 11 peuvent fonctionner en mode compatibilité ASCII. Pour plus d'informations, reportez-vous à la section Codes ASCII.

Note de compatibilité : 4D peut fonctionner avec deux ensembles de caractères : Unicode ou ASCII. Le mode Unicode est le mode standard utilisé dans les nouvelles bases à compter de la version 11 de 4D. Le mode ASCII est conservé par compatibilité dans les bases créées avec une version antérieure de 4D. Ce mode est appelé **mode compatibilité ASCII**.
Il est possible d'activer le mode Unicode dans les bases de données converties via le sélecteur Mode Unicode des commandes Lire parametre base et FIXER PARAMETRE BASE ou via l'option **Mode Unicode** placée dans la page Application/Compatibilité des Préférences :



Dans la plupart des cas, le fonctionnement initial des applications n'est pas affecté par ce paramétrage, 4D se chargeant en interne des conversions de caractères nécessaires. En outre, les caractères les plus courants (a-z, 0-9, etc...) ont la même valeur (de 1 à 127) en Unicode et en ASCII (Windows et Mac OS).
Toutefois, certaines instructions du langage, utilisant notamment les commandes travaillant avec les chaînes de caractères, pourront nécessiter des adaptations. Par exemple, l'instruction Caractere(200) ne retournera pas la même valeur en Unicode et en ASCII.

Note : Le mode est spécifique à chaque base de données. Il est donc possible de faire cohabiter une base Unicode avec des composants non Unicode (ou inversement).

4D et les codes ASCII

Lorsque la base fonctionne en mode compatibilité ASCII, sur les deux plates-formes Mac OS et Windows, le moteur interne de base de données et le langage de 4D travaillent avec la table ASCII étendue du Macintosh. Lorsque vous saisissez des données (ajout d'enregistrements, édition de méthodes, etc.), 4D utilise le schéma interne de conversion d'Altura pour convertir les codes provenant du clavier (qui sont donc exprimés à l'aide de la table ASCII étendue Windows) en codes Macintosh. Par exemple, pour saisir le caractère “ß”, vous tapez **Alt**+0223, mais c'est le code ASCII 167 que 4D va stocker dans l'enregistrement. Ce mode de fonctionnement est totalement transparent pour l'utilisateur car lorsque vous effectuez par exemple une recherche, vous saisissez la valeur réelle à trouver dans l'éditeur de recherches. La valeur que vous tapez (**Alt**+0223) est également convertie en code ASCII 167, et la recherche aboutira.

Le même principe est appliqué lorsque vous tapez **Alt**+0223 dans l'éditeur de méthodes. Notez cependant que si vous recherchez un caractère sur la base de son code ASCII, vous devrez utiliser le code ASCII Macintosh du caractère.
Par exemple :

```
CHERCHER (...; [MaTable]MonChamp="ß") ` ß s'obtient par Alt+0223
```

est identique à :

```
CHERCHER (...;[MaTable]MonChamp=Caractere(167)) `  ß a pour code ASCII Mac OS 167
```

Tables des codes ASCII

• La table standard des codes ASCII (de 0 à 127) est identique sur les plates-formes Windows et Mac OS.

• La table ASCII étendue (codes ASCII de 128 à 255) est différente entre Windows et Mac OS. Afin d'assurer l'indépendance de plate-forme de vos applications, 4D, lorsque le programme fonctionne sous Windows, convertit automatiquement les codes ASCII (de la table Windows vers la table Mac OS) lorsque des caractères sont entrés dans l'environnement 4D (saisie de données, copier/coller, import d'enregistrements, etc.) ou encore (de la table Mac OS vers la table Windows) lorsque des caractères sont extraits de l'environnement 4D (couper ou copier, export, etc.).

• Codes ASCII de 0 à 63

Car	Dec	Hex	Car	Dec	Hex
Nul	0	0	SP	32	20
SOH	1	1	!	33	21
STX	2	2	"	34	22
ETX	3	3	#	35	23
EOT	4	4	$	36	24
ENQ	5	5	%	37	25
ACK	6	6	&	38	26
BEL	7	7	'	39	27
BS	8	8	(	40	28
HT	9	9	)	41	29
LF	10	A	*	42	2A
VT	11	B	+	43	2B
FF	12	C	,	44	2C
CR	13	D	-	45	2D
SO	14	E	.	46	2E
SI	15	F	/	47	2F
DLE	16	10	0	48	30
DC1	17	11	1	49	31
DC2	18	12	2	50	32
DC3	19	13	3	51	33
DC4	20	14	4	52	34
NAK	21	15	5	53	35
SYN	22	16	6	54	36
ETB	23	17	7	55	37
CAN	24	18	8	56	38
EM	25	19	9	57	39
SUB	26	1A	:	58	3A
ESC	27	1B	;	59	3B
FS	28	1C	<	60	3C
GS	29	1D	=	61	3D
RS	30	1E	>	62	3E
US	31	1F	?	63	3F

• Codes ASCII de 64 à 127

Car	Dec	Hex	Car	Dec	Hex
@	64	40	`	96	60
A	65	41	a	97	61
B	66	42	b	98	62
C	67	43	c	99	63
D	68	44	d	100	64
E	69	45	e	101	65
F	70	46	f	102	66
G	71	47	g	103	67
H	72	48	h	104	68
I	73	49	i	105	69
J	74	4A	j	106	6A
K	75	4B	k	107	6B
L	76	4C	l	108	6C
M	77	4D	m	109	6D
N	78	4E	n	110	6E
O	79	4F	o	111	6F
P	80	50	p	112	70
Q	81	51	q	113	71
R	82	52	r	114	72
S	83	53	s	115	73
T	84	54	t	116	74
U	85	55	u	117	75
V	86	56	v	118	76
W	87	57	w	119	77
X	88	58	x	120	78
Y	89	59	y	121	79
Z	90	5A	z	122	7A
[	91	5B	{	123	7B
\	92	5C	|	124	7C
]	93	5D	}	125	7D
^	94	5E	~	126	7E
_	95	5F	Del	127	7F

• Codes ASCII de 128 à 191

Table ASCII étendue Macintosh (traitement interne 4D)			Correspondance codes ASCII étendus Windows	
Dec	Hex	Caractère (Times)	Combinaison Alt +	Caractère (Arial)
128	80	Ä	0196	Ä
129	81	Å	0197	Å
130	82	Ç	0199	Ç
131	83	É	0201	É
132	84	Ñ	0209	Ñ
133	85	Ö	0214	Ö
134	86	Ü	0220	Ü
135	87	á	0225	á
136	88	à	0224	à
137	89	â	0226	â
138	8A	ä	0228	ä
139	8B	ã	0227	ã
140	8C	å	0229	å
141	8D	ç	0231	ç
142	8E	é	0233	é
143	8F	è	0232	è
144	90	ê	0234	ê
145	91	ë	0235	ë
146	92	í	0237	í
147	93	ì	0236	ì
148	94	î	0238	î
149	95	ï	0239	ï
150	96	ñ	0241	ñ
151	97	ó	0243	ó
152	98	ò	0242	ò
153	99	ô	0244	ô
154	9A	ö	0246	ö
155	9B	õ	0245	õ
156	9C	ú	0250	ú
157	9D	ù	0249	ù
158	9E	û	0251	û
159	9F	ü	0252	ü

160	A0	†	0134	†
161	A1	°	0176	°
162	A2	¢	0162	¢
163	A3	£	0163	£
164	A4	§	0167	§
165	A5	•	0149	•
166	A6	¶	0182	¶
167	A7	ß	0223	ß
168	A8	®	0174	®
169	A9	©	0169	©
170	AA	™	0153	™
171	AB	´	0145	´
172	AC	¨	0168	¨
173	AD	≠		
174	AE	Æ	0198	Æ
175	AF	Ø	0216	Ø

176	B0	∞		
177	B1	±	0177	±
178	B2	≤		
179	B3	≥		
180	B4	¥	0165	¥
181	B5	µ	0181	µ
182	B6	∂		
183	B7	∑		
184	B8	∏		
185	B9	π		
186	BA	∫		
187	BB	ª	0170	ª
188	BC	º	0186	º
189	BD	Ω		
190	BE	æ	0230	æ
191	BF	ø	0248	ø

• Codes ASCII de 192 à 255

Table ASCII étendue Macintosh (traitement interne 4D)			Correspondance codes ASCII étendus Windows	
Dec	Hex	Caractère (Times)	Combinaison Alt +	Caractère (Arial)
192	C0	¿	0191	¿
193	C1	¡	0161	¡
194	C2	¬	0172	¬
195	C3	√		
196	C4	ƒ	0131	ƒ
197	C5	≈		
198	C6	Δ		
199	C7	«	0171	«
200	C8	»	0187	»
201	C9	…	0133	…
202	CA	(espace)	0160	(espace)
203	CB	À	0192	À
204	CC	Ã	0195	Ã
205	CD	Õ	0213	Õ
206	CE	Œ	0140	Œ
207	CF	œ	0156	œ
208	D0	–	0150	–
209	D1	—	0151	—
210	D2	“	0147	“
211	D3	”	0148	”
212	D4	‘	0145	‘
213	D5	’	0146	’
214	D6	÷	0247	÷
215	D7	◊		
216	D8	ÿ	0255	ÿ
217	D9	Ÿ	0159	Ÿ
218	DA	⁄		
219	DB	€	0164	¤
220	DC	‹	0139	‹
221	DD	›	0155	›
222	DE	fi		
223	DF	fl		

224	E0	‡	0135	‡
225	E1	·	0183	·
226	E2	‚		
227	E3	„	0132	„
228	E4	‰	0137	‰
229	E5	Â	0194	Â
230	E6	Ê	0202	Ê
231	E7	Á	0193	Á
232	E8	Ë	0203	Ë
233	E9	È	0200	È
234	EA	Í	0205	Í
235	EB	Î	0206	Î
236	EC	Ï	0207	Ï
237	ED	Ì	0204	Ì
238	EE	Ó	0211	Ó
239	EF	Ô	0212	Ô
240	F0			
241	F1	Ò	0210	Ò
242	F2	Ú	0218	Ú
243	F3	Û	0219	Û
244	F4	Ù	0217	Ù
245	F5	ı	0185	ı
246	F6	ˆ		
247	F7	˜	0152	~
248	F8	¯	0175	¯
249	F9	˘		
250	FA	˙		
251	FB	˚		
252	FC	¸	0184	¸
253	FD	˝		
254	FE	˛		
255	FF	ˇ		

Note : Les cases grisées signalent des caractères non disponibles sous Windows, ou différents des caractères Macintosh.

Référence

APPELER SUR EVENEMENT, Code de caractere, ISO vers Mac, Mac vers ISO, Mac vers Windows, Windows vers Mac.

4D retourne le code clavier des touches de fonction dans la variable système Keycode, qui est utilisée dans les méthodes projet installées par la commande APPELER SUR EVENEMENT pour intercepter les événements.

Les valeurs des touches de fonctions ne sont pas basées sur des codes ASCII. Elles sont listées ci-dessous :

Touche	Code
F1	-122
F2	-120
F3	-99
F4	-118
F5	-96
F6	-97
F7	-98
F8	-100
F9	-101
F10	-109
F11	-103
F12	-111
F13	-105
F14	-107
F15	-113

Par ailleurs, le tableau suivant liste les valeurs retournées dans la variable système Keycode lorsque vous appuyez sur les touches standard comme Retour chariot ou Entrée.

Touche	Code
Entrée	3
Retour chariot	13
Retour arrière	8
Tabulation	9
Echappement	27
Effacement	127
Aide	5

Début	1
Fin	4
Haut de page	11
Bas de page	12
Flèche gauche	28
Flèche droite	29
Flèche haut	30
Flèche bas	31

Référence

APPELER SUR EVENEMENT.

71

Syntaxe des commandes

La première colonne indique le numéro de chaque commande, utilisé notamment par la commande Nom commande.

A

99 Abs (nombre) → Numérique
346 Activation → Booléen
192 ACTIVER BOUTON ({*; }objet)
149 ACTIVER LIGNE MENU (menu; ligneMenu{; process})
119 ADJOINDRE ELEMENT ({table; }ensemble)
431 AFFICHER BARRE DE MENUS
433 AFFICHER BARRE OUTILS
105 AFFICHER ENREGISTREMENT {(table)}
435 AFFICHER FENETRE {(fenêtre)}
910 AFFICHER NOTIFICATION (titre; contenu{; délai})
393 Ajouter a date (date; années; mois; jours) → Date
265 Ajouter a document (document{; type}) → DocRef
376 AJOUTER A LISTE (liste; texteElément; réfElément{; sous_Liste; déployée})
911 AJOUTER A TABLEAU (tableau; valeur)
403 AJOUTER DONNEES AU CONTENEUR (typeDonnées; données)
56 AJOUTER ENREGISTREMENT ({table}{; }{*})
411 AJOUTER LIGNE MENU (menu; libelléLigne{; sousMenu}{; process}{; *})
361 AJOUTER SEGMENT DE DONNEES
202 AJOUTER SOUS ENREGISTREMENT (sousTable; formulaire{; *})
41 ALERTE (message{; libelléBoutonOK})
206 ALLER A CHAMP ({*; }objet)
200 ALLER A DERNIER ENREGISTREMENT {(table)}
201 ALLER A DERNIER SOUS ENREGISTREMENT (sousTable)
242 ALLER A ENREGISTREMENT ({table; }enregistrement)
247 ALLER A PAGE (numéroPage)
245 ALLER DANS SELECTION ({table; }position)
35 Ancien (champ) → Expression
263 ANCIEN LIEN RETOUR (champ)
25 Annee de (date) → Numérique
241 ANNULER TRANSACTION
273 Appartient a ensemble (ensemble) → Booléen
338 Appartient au groupe (nomUtilisateur; groupe) → Booléen
328 Appel exterieur → Booléen
311 APPELER 4D
329 APPELER PROCESS (process)

316 APPELER SUR A PROPOS (libelléLigne; méthode)
155 APPELER SUR ERREUR (méthodErreur)
190 APPELER SUR EVENEMENT (méthodeEvén{; nomProcess})
778 APPELER WEB SERVICE (urlAccès; soapAction; nomMéthode; espaceNommage{; typeComposé{; *}})
70 APPLIQUER A SELECTION (laTable; formule)
73 APPLIQUER A SOUS SELECTION (sousTable; formule)
882 APPLIQUER TRANSFORMATION XSLT (sourceXML; feuilleXSL; résultat{; compileFeuille})
31 Apres → Booléen
20 Arctan (nombre) → Numérique
963 ARRETER SERVEUR SQL
618 ARRETER SERVEUR WEB
94 Arrondi (nombre; nbDécimales) → Numérique
366 ASSOCIER TYPES FICHIER (macOS; windows; contexte)
786 AUTHENTIFIER WEB SERVICE (nom; motDePasse{; méthodeAuth}{; *})
29 Avant → Booléen
198 Avant selection {(table)} → Booléen
199 Avant sous enregistrement (sousTable) → Booléen

B

151 BEEP
526 BLOB VERS DOCUMENT (document; blob{; *})
549 BLOB vers entier (blob; ordreOctet{; offset}) → Numérique
551 BLOB vers entier long (blob; ordreOctet{; offset}) → Numérique
682 BLOB VERS IMAGE (blobImage; image{; codec})
557 BLOB vers liste (blob{; offset}) → RefList
553 BLOB vers reel (blob; formatRéel{; offset}) → Numérique
555 BLOB vers texte (blob; formatTexte{; offset{; longueurTexte}}) → Chaîne
850 BLOB VERS UTILISATEURS (utilisateurs)
533 BLOB VERS VARIABLE (blob; variable{; offset})

C

432 CACHER BARRE DE MENUS
434 CACHER BARRE OUTILS
436 CACHER FENETRE {(fenêtre)}
324 CACHER PROCESS (process)
90 Caractere (codeCaractère) → Chaîne
10 Chaine (expression{; format}) → Chaîne
180 Chaine heure (secondes) → Alpha
253 Champ (numTable| ptrChamp{; numChamp}) → Num | Pointeur
444 CHANGER COORDONNEES FENETRE (gauche; haut; droite; bas{; fenêtre})
531 CHANGER CREATEUR DOCUMENT (document; créateur)
904 CHANGER DICTIONNAIRE (dictionnaire)

385 CHANGER ELEMENT({*; } liste; réfElément | *; textElément; nouvelRéf{; sous_Liste; déployée})
164 CHANGER JEU DE CARACTERES ({*; }objet; police)
637 CHANGER LICENCES
186 CHANGER MOT DE PASSE (motDePasse)
469 CHANGER POINTEUR SOURIS {(curseur)}
482 CHANGER POSITION DANS DOCUMENT (docRef; offset{; ancre})
281 CHANGER PRIVILEGES
478 CHANGER PROPRIETES DOCUMENT (document; verrouillé; invisible; créé le; créé à; modifié le; modifié à)
386 CHANGER PROPRIETES ELEMENT ({*; }liste; réfElément | *; saisissable; styles; icône{; couleur})
387 CHANGER PROPRIETES LISTE (liste; apparence{; icône{; hauteurLigne{; doubleClic{; multiSélection{; modifiable}}}}})
166 CHANGER STYLE ({*; }objet; style)
165 CHANGER TAILLE ({*; }objet; taille)
480 CHANGER TAILLE DOCUMENT (document; taille)
213 CHANGER TITRE FENETRE (titre{; fenêtre})
530 CHANGER TYPE DOCUMENT (document; type)
289 CHANGER UTILISATEUR COURANT{(utilisateur; motDePasse)}
44 CHARGER ANCIEN (champ)
52 CHARGER ENREGISTREMENT {(table)}
185 CHARGER ENSEMBLE ({table; }ensemble; document)
357 CHARGER ET COMPRESSER IMAGE (document; type; qualité; image)
383 Charger liste (nomListe) → RéfListe
42 CHARGER SUR LIEN (tableN | champN{; discriminant})
277 CHERCHER ({table}{; critèreRecherche{; *}})
952 Chercher dans liste ({*; }liste; valeur; portée{; tabEléments{; *}}) → Entier long
341 CHERCHER DANS SELECTION ({laTable}{; critère{; *}})
230 Chercher dans tableau (tableau; valeur{; début}) → Numérique
449 Chercher fenetre (gauche; haut{; partieFenêtre}) → RefFen
292 CHERCHER PAR EXEMPLE ({table}{; }{*})
48 CHERCHER PAR FORMULE (table{; formule})
207 CHERCHER PAR FORMULE DANS SELECTION (laTable{; formule})
942 CHERCHER PAR SQL ({uneTable; }formuleSQL)
644 CHERCHER PAR TABLEAU (champCible; tableau)
1050 CHERCHER PAR TABLEAU DANS SELECTION (champCible; tableau)
372 Chercher process (nom{; *}) → Numérique
108 CHERCHER SOUS ENREGISTREMENTS (sousTable; formule)
955 Choisir (critère; valeur{; valeur2; ...; valeurN}) → Expression
271 CHOIX COULEUR ({*; }objet; couleur{; couleurAlt})
237 CHOIX ENUMERATION ({*; }objet; énum)
235 CHOIX FILTRE SAISIE ({*; }objet; filtreSaisie)
236 CHOIX FORMATAGE ({*; }objet; formatAffich)
238 CHOIX SAISISSABLE ({*; }objet; zoneSaisie)

603 CHOIX VISIBLE ({*; }objet; visible)
843 CHOIX VISIBLE BARRES DEFILEMENT ({*; }objet; horizontal; vertical)
713 Clic contextuel → Booléen
712 Clic droit → Booléen
91 Code de caractere (unCaractère) → Numérique
987 COMBINER IMAGES (imageRésultat; image1; opérateur; image2{; décalHoriz; décalVert})
937 Compacter fichier donnees (cheminStructure; cheminDonnées; dossierArchive; options; méthode) → Texte
534 COMPRESSER BLOB (blob{; compression})
359 COMPRESSER FICHIER IMAGE (document; type; qualité)
355 COMPRESSER IMAGE (image; type; qualité)
907 Compter dans tableau (tableau; valeur) → Entier long
162 CONFIRMER (message{; libelléBoutonOK{; libelléBoutonAnn}})
698 Connexion Web securisee → Booléen
657 Contexte Web → Booléen
360 Convertir caracteres (chaîne; target; srcMask) → Chaîne
1011 CONVERTIR DEPUIS TEXTE (texte4D; jeuCaractères; blobConverti)
1002 CONVERTIR IMAGE (image; codec)
1012 Convertir vers texte (blob; jeuCaractères) → Texte
676 Convertisseur Euro (valeur; deMonnaie; versMonnaie) → Numérique
438 COORDONNEES ECRAN (gauche; haut; droite; bas{; écran})
443 COORDONNEES FENETRE (gauche; haut; droite; bas{; fenêtre})
558 COPIER BLOB (srcBLOB; dstBLOB; srcOffset; dstOffset; nombre)
541 COPIER DOCUMENT (nomSource; nomDest{; *})
600 COPIER ENSEMBLE (srcEns; dstEns)
626 Copier liste (liste) → RéfListe
331 COPIER SELECTION ({table; }tempo)
226 COPIER TABLEAU (source; destination)
900 CORRECTION ORTHOGRAPHIQUE
18 Cos (nombre) → Numérique
529 Createur document (document) → Alpha
694 CREER ALIAS (cheminCible; cheminAlias)
266 Creer document (document{; leType}) → DocRef
475 CREER DOSSIER (cheminAccès)
68 CREER ENREGISTREMENT {(table)}
641 CREER ENSEMBLE SUR TABLEAU (table; tabEnrg{; nomEns})
153 Creer fenetre (gauche; haut; droite; bas{; type{; titre{; caseFermeture}}}){ → RefFen }
309 Creer fenetre externe (gauche; haut; droit; bas; type; titre; zonePlugin) → Numérique
675 Creer fenetre formulaire ({laTable; }nomForm{; type{; posH{; posV{; *}}}}) → RefFen
313 CREER FICHIER DONNEES (cheminAccès)
496 Creer fichier ressources (resNomFichier{; typeFichier{; *}}) → DocRef
808 CREER FORMULAIRE UTILISATEUR (table; formulaire; formUtilisateur)
679 CREER IMAGETTE (source; dest{; largeur{; hauteur{; mode{; profondeur}}}})
966 CREER INDEX (laTable; tabChamps; typeIndex; nomIndex{; *})

408 Creer menu {(menu)} → RefMenu
640 CREER SELECTION SUR TABLEAU (table; tabEnrg{; tempo})
72 CREER SOUS ENREGISTREMENT (sousTable)
65 CREER SUR LIEN (champ)
689 CRYPTER BLOB (aCrypter; cléPrivEmetteur{; cléPubRécepteur})
303 CUMULER SUR (objet{; objet2; ...; objetN})
293 C_ALPHA ({méthode; }taille; variable{; variable2; ...; variableN})
604 C_BLOB ({méthode; }variable{; variable2; ...; variableN})
305 C_BOOLEEN ({méthode; }variable{; variable2; ...; variableN})
307 C_DATE ({méthode; }variable{; variable2; ...; variableN})
282 C_ENTIER ({méthode; }variable{; variable2; ...; variableN})
283 C_ENTIER LONG ({méthode; }variable{; variable2; ...; variableN})
352 C_GRAPHE ({méthode; }variable{; variable2; ...; variableN})
306 C_HEURE ({méthode; }variable{; variable2; ...; variableN})
286 C_IMAGE ({méthode; }variable{; variable2; ...; variableN})
301 C_POINTEUR ({méthode; }variable{; variable2; ...; variableN})
285 C_REEL ({méthode; }variable{; variable2; ...; variableN})
284 C_TEXTE ({méthode; }variable{; variable2; ...; variableN})

D

102 Date (chaîneDate) → Date
33 Date du jour {(*)} → Date
50 DEBUT SELECTION {(table)}
61 DEBUT SOUS ENREGISTREMENT (sousTable)
948 Debut SQL
239 DEBUT TRANSACTION
9 Dec (nombre) → Numérique
782 DECLARATION SOAP (variable; type; entrée_sortie{; alias})
896 DECODER (blob)
535 DECOMPRESSER BLOB (blob)
1044 DECRIRE EXECUTION RECHERCHE (statut)
690 DECRYPTER BLOB (aDécrypter; cléPubEmetteur{; cléPrivRécepteur})
906 DEFILER LIGNES ({*; }objet{; position{; *}})
163 Demander (message{; réponseDéfaut{; titreBoutonOK{; titreBoutonAnn}}}) → Alpha
177 DEPILER ENREGISTREMENT {(table)}
540 DEPLACER DOCUMENT (cheminSource; cheminDest)
452 DEPLACER FENETRE
664 DEPLACER OBJET ({*; }objet; dépH; dépV{; redimH{; redimV{; *}}})
334 DEPLACER SELECTION ({table; }tempo)
251 DERNIERE PAGE
347 Desactivation → Booléen
649 DESINSCRIRE CLIENT
40 DIALOGUE ({table; }formulaire{; *})
122 DIFFERENCE (ensemble1; ensemble2; résultat)

525 DOCUMENT VERS BLOB (document; blob{; *})
719 DOM Analyser source XML (document{; validation{; dtd | schéma}}) → Chaîne
720 DOM Analyser variable XML (variable{; validation{; dtd}}) → Chaîne
864 DOM Chercher element XML (refElément; xChemin{; tabRefEléments}) → refElément
1010 DOM Chercher element XML par ID (refElément; id) → Chaîne
727 DOM Compter attributs XML (refElément) → Entier long
726 DOM Compter elements XML (refElément; nomElément) → Entier long
865 DOM Creer element XML (refElément; xChemin{; nomAttribut{; valeurAttribut}}{; nomAttribut2; valeurAttribut2; ...; nomAttributN; valeurAttributN}) → Chaîne
861 DOM Creer ref XML (racine{; nameSpace{; nSNom{; nSValeur}}}{; nSNom2; nSValeur2; ...; nSNomN; nSValeurN}) → Chaîne
866 DOM ECRIRE ATTRIBUT XML (refElément; nomAttribut; valeurAttribut{; nomAttribut2; valeurAttribut2; ...; nomAttributN; valeurAttributN})
867 DOM ECRIRE NOM ELEMENT XML (refElément; nomElément)
859 DOM ECRIRE OPTIONS XML (refElément; encodage{; autonome{; indentation}})
868 DOM ECRIRE VALEUR ELEMENT XML (refElément{; xChemin; valeurElément{; *})
862 DOM EXPORTER VERS FICHIER (refElément; cheminFichier)
863 DOM EXPORTER VERS VARIABLE (refElément; vVarXml)
722 DOM FERMER XML (refElément)
729 DOM LIRE ATTRIBUT XML PAR INDEX (refElément; indexAttribut; nomAttribut; valeurAttribut)
728 DOM LIRE ATTRIBUT XML PAR NOM (refElément; nomAttribut; valeurAttribut)
925 DOM Lire dernier element XML enfant (refElément{; nomElémentEnf{; valeurElémentEnf}}) → Chaîne
725 DOM Lire element XML (refElément; nomElément; index; valeurElément) → Chaîne
924 DOM Lire element XML frere precedent (refElément{; nomElémentFrère{; valeurElémentFrère}}) → Chaîne
724 DOM Lire element XML frere suivant (refElément{; nomElémentFrère{; valeurElémentFrère}}) → Chaîne
923 DOM Lire element XML parent (refElément{; nomElémentPar{; valeurElémentPar}}) → Chaîne
1053 DOM Lire element XML racine (refElément) → Chaîne
721 DOM Lire informations XML (refElément; infoXML) → Chaîne
730 DOM LIRE NOM ELEMENT XML (refElément; nomElément)
723 DOM Lire premier element XML enfant (refElément{; nomElémentEnf{; valeurElémentEnf}}) → Chaîne
731 DOM LIRE VALEUR ELEMENT XML (refElément; valeurElément{; cDATA})
869 DOM SUPPRIMER ELEMENT XML (refElément)
485 Dossier 4D ({dossier}{; }{*}) → Alpha
487 Dossier systeme {(type)} → Alpha
486 Dossier temporaire → Alpha
225 DUPLIQUER ENREGISTREMENT {(table)}

E

26 Ecart type (séries) → Numérique
441 Ecran principal → Entier long
845 ECRIRE ACCES PLUGIN (plugIn; groupe)
297 ECRIRE CACHE
680 ECRIRE FICHIER IMAGE (nomFichier; image{; codec})
566 ECRIRE IMAGE DANS BIBLIOTHEQUE (image; refImage; nomImage)
514 ECRIRE NOM RESSOURCE (resType; resNum; resNom{; resFichier})
614 Ecrire proprietes groupe (réfGroupe; nom; propriétaire{; membres}) → Numérique
516 ECRIRE PROPRIETES RESSOURCE (resType; resNum; resAttr{; resFichier})
612 Ecrire proprietes utilisateur (réfUtilisateur; nom; démarrage; motDePasse; nbUtilisation; dernièreUtilisation{; adhésions{; groupePropriétaire}}) → Numérique
509 ECRIRE RESSOURCE (resType; resNum; resDonnées{; resFichier})
507 ECRIRE RESSOURCE CHAINE (resNum; resDonnées{; resFichier})
503 ECRIRE RESSOURCE IMAGE (resNum; resDonnées{; resFichier})
505 ECRIRE RESSOURCE TEXTE (resNum; resDonnées{; resFichier})
370 ECRIRE VARIABLE PROCESS (process; varDestination; exprSource{; varDestination2; exprSource2; ...; varDestinationN; exprSourceN})
75 ECRIRE VARIABLES (doc; variable{; variable2; ...; variableN})
84 ECRITURE DIF ({laTable; }document)
85 ECRITURE SYLK ({laTable; }document)
870 EDITER ELEMENT ({*; }objet{; élément})
806 EDITER FORMULE (table; formule)
402 EFFACER CONTENEUR
117 EFFACER ENSEMBLE (ensemble)
160 EFFACER FENETRE {(fenêtre)}
978 EFFACER MENU (menu)
333 EFFACER SELECTION (tempo)
144 EFFACER SEMAPHORE (sémaphore)
89 EFFACER VARIABLE (variable)
633 Element parent ({*; }liste; réfElément | *) → Entier long
379 Elements selectionnes ({*; }liste{; tabEléments{; *}}) → Entier long
176 EMPILER ENREGISTREMENT {(table)}
112 En entete → Booléen
191 En pied → Booléen
113 En rupture → Booléen
895 ENCODER (blob)
323 ENDORMIR PROCESS (process; durée)
561 ENLEVER ELEMENT ({table; }ensemble)
669 Enregistrement charge {(table)} → Booléen
314 Enregistrement modifie {(table)} → Booléen
110 ENREGISTREMENT PRECEDENT {(table)}
189 ENREGISTREMENT SELECTION {(table)}
51 ENREGISTREMENT SUIVANT {(table)}

147 Enregistrement verrouille {(table)} → Booléen
195 Enregistrements dans ensemble (ensemble) → Numérique
83 Enregistrements dans table {(table)} → Numérique
76 Enregistrements trouves {(table)} → Numérique
667 ENREGISTRER EVENEMENT ({typeSortie; }message{; importance})
358 ENREGISTRER IMAGE (document; image)
140 ENSEMBLE VIDE ({table; }ensemble)
8 Ent (nombre) → Numérique
550 ENTIER LONG VERS BLOB (entierLong; blob; ordreOctets{; offset | *})
548 ENTIER VERS BLOB (entier; blob; ordreOctet{; offset | *})
288 ENUMERATION VERS TABLEAU (énumération; tableau{; réfEléments})
654 ENVOYER BLOB HTML (blob; type{; sansContexte})
815 ENVOYER DONNEES HTTP (données{; *})
78 ENVOYER ENREGISTREMENT {(table)}
781 ENVOYER ERREUR SOAP (typeErreur; description)
619 ENVOYER FICHIER HTML (fichierHTML)
103 ENVOYER PAQUET ({docRef; }paquet)
659 ENVOYER REDIRECTION HTTP (url{; *})
677 ENVOYER TEXTE HTML (texteHTML{; sansContexte})
80 ENVOYER VARIABLE (variable)
1000 Est un numero de champ valide (numTable | ptrTable; numChamp) → Booléen
999 Est un numero de table valide (numTable) → Booléen
783 Est une requete SOAP → Booléen
294 Est une variable (unPointeur) → Booléen
362 Etat lecture seulement {(table)} → Booléen
388 Evenement formulaire → Numérique
369 Evenement moteur → Entier long
63 EXECUTER FORMULE (instruction)
1007 EXECUTER METHODE (nomMéthode; résultat | *{; param}{; param2; ...; paramN})
651 EXECUTER SUR CLIENT (nomClient; nomMéthode{; param}{; param2; ...; paramN})
373 Executer sur serveur (procédure; pile{; nom{; param{; param2; ...; paramN}{; *}}}) → Numérique
21 Exp (nombre) → Numérique
666 EXPORTER DONNEES (nomFichier{; projet{; *}})
167 EXPORTER TEXTE ({laTable; }document)

F

215 Faux → Booléen
827 Fenetre formulaire courant → RefFen
447 Fenetre premier plan {(*)} → RefFen
448 Fenetre suivante (fenêtre) → RefFen
267 FERMER DOCUMENT (réfDocument)
154 FERMER FENETRE {(fenêtre)}
498 FERMER FICHIER RESSOURCES (resFichier)

996 FERMER TACHE IMPRESSION
491 Fichier application → Alpha
490 Fichier donnees {(segment)} → Alpha
716 Fichier donnees verrouille → Booléen
928 Fichier historique → Alpha
489 Fichier structure {(*)} → Alpha
321 FILTRER EVENEMENT
389 FILTRER FRAPPE CLAVIER (carFiltré)
36 Fin de selection {(table)} → Booléen
37 Fin sous enregistrement (sousTable) → Booléen
949 Fin SQL
706 FIXER ALIGNEMENT ({*; }objet; alignement)
364 FIXER APERCU IMPRESSION (aperçu)
67 FIXER BARRE MENUS (barre{; process}{; *})
842 FIXER COULEUR GRILLE LISTBOX ({*; }objet; couleur; horizontal; vertical)
628 FIXER COULEURS RVB ({*; }objet; couleurAvantPlan; couleurArrièrePlan{; couleurArrièrePlanAlt})
396 FIXER DESTINATION RECHERCHE (destinationType{; destinationObjet})
660 FIXER ENTETE HTTP (entête|tabChamps{; tabValeurs})
813 FIXER EXECUTABLE CGI (url1{; url2})
975 FIXER FICHIER DANS CONTENEUR (cheminFichier)
345 FIXER FICHIER HISTORIQUE (historique | *)
835 FIXER HAUTEUR LIGNES LISTBOX ({*; }objet; hauteur)
950 FIXER ICONE ELEMENT ({*; }liste; refElément | *; icône)
984 FIXER ICONE LIGNE MENU (menu; ligneMenu; refIcône{; process})
521 FIXER IMAGE DANS CONTENEUR (image)
787 FIXER IMPRIMANTE COURANTE (nomImpr)
344 FIXER INDEX (champ; index{; mode}{; *})
367 FIXER INTERFACE (interface)
833 FIXER LARGEUR COLONNE LISTBOX ({*; }objet; largeur)
919 FIXER LIEN CHAMP (tableN | champN; aller; retour)
310 FIXER LIENS AUTOMATIQUES (aller{; retour})
395 FIXER LIMITE RECHERCHE (limite)
620 FIXER LIMITES AFFICHAGE WEB (nombreEnr{; nombrePages{; imageRéf}})
710 FIXER MARGE IMPRESSION (gauche; haut; droite; bas)
208 FIXER MARQUE LIGNE MENU (menu; ligneMenu; marque{; process})
982 FIXER METHODE LIGNE MENU (menu; ligneMenu; nomMéthode{; process})
805 FIXER METHODES AUTORISEES (tabMéthodes)
645 FIXER MINUTEUR (tickCount)
623 FIXER NIVEAU COMPARAISON REEL (epsilon)
733 FIXER OPTION IMPRESSION (option; valeur1{; valeur2})
901 FIXER OPTION WEB SERVICE (option; valeur)
639 FIXER PAGE ACCUEIL (homePage)
642 FIXER PARAMETRE BASE ({table; }sélecteur; valeur)
986 FIXER PARAMETRE ELEMENT ({*; }liste; réfElément | *; sélecteur; valeur)

1004 FIXER PARAMETRE LIGNE MENU (menu; ligneMenu; param)
998 FIXER PARAMETRE MACRO (sélecteur; paramTexte)
777 FIXER PARAMETRE WEB SERVICE (nom; valeur{; typeSOAP})
883 FIXER PARAMETRE XSLT (nomParam; valeurParam)
953 FIXER POLICE ELEMENT ({*; }liste; réfElément | *; police)
537 FIXER PROFONDEUR ECRAN (profondeur{; couleurs{; écran}})
973 FIXER PROPRIETE LIGNE MENU (menu; ligneMenu; propriété; valeur{; process})
423 FIXER RACCOURCI LIGNE MENU (menu; ligneMenu; touche{; modificateurs}{; process})
634 FIXER RACINE HTML (dossierRacine)
661 FIXER RECHERCHE ET VERROUILLAGE (verrou)
892 FIXER REDIMENSIONNEMENT FORMULAIRE HORIZONTAL (redimension{; largeurMini{; largeurMaxi}})
893 FIXER REDIMENSIONNEMENT FORMULAIRE VERTICAL (redimension{; hauteurMini{; hauteurMaxi}})
425 FIXER STYLE LIGNE MENU (menu; ligneMenu; styleLigne{; process})
1013 FIXER TABLE SOURCE LISTBOX ({*; }objet; numTable | tempo)
606 FIXER TAILLE BLOB (blob; taille{; remplisseur})
891 FIXER TAILLE FORMULAIRE ({objet; }horizontal; vertical{; *})
709 FIXER TAQUET IMPRESSION (numTaquet; position{; *})
622 FIXER TEMPORISATION WEB (timeout)
523 FIXER TEXTE DANS CONTENEUR (texte)
348 FIXER TEXTE LIGNE MENU (menu; ligneMenu; texteLigne{; process}{; *})
268 FIXER TIMEOUT (secondes)
602 FIXER TITRES CHAMPS (table | sous-table; titresChamps; numChamps{; *})
601 FIXER TITRES TABLES (titresTables; numTables{; *})
965 FIXER VALEUR CHAMP NULL (unChamp)
812 FIXER VARIABLE ENVIRONNEMENT (nomVar; valeurVar)
55 FORMULAIRE ENTREE ({laTable; }formulaire{; formUtilisateur{; *}})
54 FORMULAIRE SORTIE ({laTable; }formulaire{; formUtilisateur})
390 Frappe clavier → Alpha

G

871 GENERER APPLICATION {(nomProjet)}
688 GENERER CLES CRYPTAGE (cléPrivée; cléPublique{; longueur})
466 GENERER CLIC (sourisX; sourisY{; process{; *}})
691 GENERER DEMANDE CERTIFICAT (cléPrivée; demCertif; tabCodes; tabLibellés)
467 GENERER EVENEMENT (quoi; message; quand; sourisX; sourisY; modifiers{; process})
465 GENERER FRAPPE CLAVIER (code{; modifiers{; process}})
488 Gestalt (sélecteur; valeur) → Numérique
169 GRAPHE (zoneGraphe; graphNum; xCatégories; zValeurs{; zValeurs2; ...; zValeursN})
148 GRAPHE SUR SELECTION ({table; }numGraphe; axeX; axeZ{; axeZ2; ...; axeZN})

H

100 Hasard → Numérique
440 Hauteur barre de menus → Entier long
1016 Hauteur barre outils → Entier long
188 Hauteur ecran {(*)} → Numérique
179 Heure (chaineHeure) → Heure
178 Heure courante {(*)} → Heure

I

692 IMAGE VERS BLOB (image; blobImage; codec)
671 IMAGE VERS GIF (imagePICT; blobGIF)
665 IMPORTER DONNEES (nomFichier{; projet{; *}})
168 IMPORTER TEXTE ({laTable; }document)
71 IMPRIMER ENREGISTREMENT ({table}{; }{* | >})
39 IMPRIMER ETIQUETTES ({table}{; étiqFichier{; * | >}})
5 Imprimer ligne ({laTable; }formulaire{; zone1{; zone2}}){ → Numérique }
60 IMPRIMER SELECTION ({table}{; }{* | >})
193 INACTIVER BOUTON ({*; }objet)
150 INACTIVER LIGNE MENU (menu; ligneMenu{; process}{; *})
82 Indefinie (variable) → Booléen
378 INFORMATION ELEMENT ({*; }liste; positionElém | *; réfElément; textElém{; sous_Liste{; déployé}})
336 INFORMATIONS PROCESS (process; procNom; procStatut; procTemps{; procVisible{; uniqueID{; origine}}})
648 INSCRIRE CLIENT (nomClient{; période{; *}})
231 Inserer chaine (source; insertion; position) → Alpha
970 INSERER COLONNE FORMULE LISTBOX ({*; }objet; positionCol; nomCol; formule; typeDonnées; nomEntête; variableEntête)
829 INSERER COLONNE LISTBOX ({*; }objet; positionCol; nomCol; variableCol; nomEntête; variableEntête)
559 INSERER DANS BLOB (blob; offset; nombre{; défaut})
625 INSERER DANS LISTE({*; } liste; avantElément | *; texteElément; réfElément{; sous_Liste; déployé})
227 INSERER DANS TABLEAU (tableau; positionDépart{; combien})
913 INSERER LIGNE LISTBOX ({*; }objet; positionLigne)
412 INSERER LIGNE MENU (menu; aprèsLigne; libelléLigne{; sousMenu}{; process})
927 INTEGRER FICHIER HISTORIQUE (cheminAccès)
121 INTERSECTION (ensemble1; ensemble2; résultat)
93 INVERSER FOND ({*; }textVar | textChp)
520 ISO vers Mac (texte) → Chaîne

J

349 JOINTURE (tableN; table1)
290 JOUER SON (nomObjet{; canal})
23 Jour de (date) → Numérique

L

181 LAISSER MESSAGES
811 LANCER PROCESS EXTERNE (nomFichier{; fluxEntrée{; fluxSortie{; fluxErreur}}})
962 LANCER SERVEUR SQL
617 LANCER SERVEUR WEB
187 Largeur ecran {(*)} → Numérique
86 LECTURE DIF ({laTable; }document)
146 LECTURE ECRITURE {(table | *)}
145 LECTURE SEULEMENT {(table | *)}
87 LECTURE SYLK ({laTable; }document)
212 LIBERER ENREGISTREMENT {(table)}
714 Licence disponible {(licence)} → Booléen
262 LIEN RETOUR (table1 | champ1)
846 Lire acces plugin (plugIn) → Alpha
707 Lire alignement ({*; }objet) → Numérique
510 Lire chaine dans liste (resNum; strNum{; resFichier}) → Chaîne
650 LIRE CLIENTS INSCRITS (listeClients; nbMéthodes)
814 LIRE CORPS HTTP (corps)
909 LIRE CORRESPONDANCE PORT SERIE (tabNums; tabNoms)
1045 Lire dernier chemin recherche (formatDesc) → Chaîne
1046 Lire dernier plan recherche (formatDesc) → Chaîne
401 LIRE DONNEES CONTENEUR (typeDonnées; données)
902 LIRE ENREGISTREMENTS MARQUES ({table; }nomEnsemble)
697 LIRE ENTETE HTTP (entête|tabChamps{; tabValeurs})
732 LIRE ERREUR XML (refElément; texteErreur{; ligne{; colonne}})
884 LIRE ERREUR XSLT (texteErreur{; ligne{; colonne}})
976 Lire fichier dans conteneur (indiceN) → Chaîne
678 LIRE FICHIER IMAGE (nomFichier; image{; *})
894 Lire formatage ({*; }objet) → Chaîne
994 LIRE FORMATAGE SYSTEME (formatage; valeur)
702 Lire hauteur imprimee → Numérique
836 Lire hauteur lignes listbox ({*; }objet) → Entier
700 LIRE ICONE DOCUMENT (cheminDoc; icône{; taille})
951 LIRE ICONE ELEMENT ({*; }liste; réfElément | *; icône)
983 LIRE ICONE LIGNE MENU (menu; ligneMenu; refIcône{; process})
699 Lire ID ressource composant (nomComp; resType; resNumOriginal) → Numérique
565 LIRE IMAGE DANS BIBLIOTHEQUE (refImage | nomImage; image)
522 LIRE IMAGE DANS CONTENEUR (image)

788 Lire imprimante courante → Chaîne
917 Lire information listbox ({*; }objet; info) → Entier long
889 LIRE INFORMATION RESTITUTION (sélecteur; info1; info2)
888 LIRE INFORMATION SAUVEGARDE (sélecteur; info1; info2)
696 LIRE INFORMATIONS SERIALISATION (clé; utilisateur; société; connectés; maxUtilisateurs)
784 Lire informations SOAP (numInfo) → Chaîne
780 Lire infos erreur Web Service (typeInfo) → Chaîne
470 Lire interface → Numérique
1009 Lire langue courante base → Chaîne
834 Lire largeur colonne listbox ({*; }objet) → Entier
920 LIRE LIEN CHAMP (champN; aller; retour{; *})
899 LIRE LIENS AUTOMATIQUES (aller; retour)
977 LIRE LIGNES MENU (menu; tabTitresMenus; tabRefsMenus)
610 LIRE LISTE GROUPE (nomsGroupe; numérosGroupe)
609 LIRE LISTE UTILISATEURS (nomsUtil; réfUtil)
711 LIRE MARGE IMPRESSION (gauche; haut; droite; bas)
428 Lire marque ligne menu (menu; ligneMenu{; process}) → Alpha
981 Lire methode ligne menu (menu; ligneMenu{; process}) → Chaîne
908 LIRE METHODES AUTORISEES (tabMéthodes)
980 Lire modificateurs ligne menu (menu; ligneMenu{; process}) → Nombre
513 Lire nom ressource (resType; resNum{; resFichier}) → Alpha
831 Lire nombre colonnes listbox ({*; }objet) → Entier long
915 Lire nombre lignes listbox ({*; }objet) → Entier long
255 Lire numero dernier champ (numTable | ptrTable) → Numérique
254 Lire numero derniere table → Numérique
898 LIRE OBJETS FORMULAIRE (tabObjets{; tabVariables{; tabPages{; *}}})
734 LIRE OPTION IMPRESSION (option; valeur1{; valeur2})
643 Lire parametre base ({table; }sélecteur{; valeurAlpha}) → Entier long
985 LIRE PARAMETRE ELEMENT ({*; }liste; réfElément | *; sélecteur; valeur)
969 LIRE PARAMETRE FORMULAIRE ({table; }formulaire; sélecteur; valeur)
1003 Lire parametre ligne menu (menu; ligneMenu) → Chaîne
1005 Lire parametre ligne menu selectionnee → Chaîne
997 LIRE PARAMETRE MACRO (sélecteur; paramTexte)
1015 LIRE PILE DERNIERE ERREUR (tabCodes; tabCompInternes; tabLibellés)
954 Lire police element ({*; }liste; réfElément | *) → Chaîne
971 LIRE POSITION CELLULE LISTBOX ({*; }objet; colonne; ligne{; varCol})
972 LIRE PROPRIETE LIGNE MENU (menu; ligneMenu; propriété; valeur{; process})
536 LIRE PROPRIETES BLOB (blob; compressé{; tailleDécompressée{; tailleCourante}})
258 LIRE PROPRIETES CHAMP (chpPtr | tableNum{; champNum; champType{; champLong{; indexé{; unique{; invisible}}}}})
631 LIRE PROPRIETES ELEMENT ({*; }liste; réfElément | *; saisissable{; styles{; icône{; couleur}}})
674 LIRE PROPRIETES FORMULAIRE ({table; }nomForm; largeur; hauteur{; nbPages{; largeurFixe{; hauteurFixe{; titre}}}})

613 LIRE PROPRIETES GROUPE (réfGroupe; nom; propriétaire{; membres})
686 LIRE PROPRIETES LIEN (ptrChp|numTable{; numChamp; tableDest; champDest{; discriminant{; allerAuto{; retourAuto}}}})
632 LIRE PROPRIETES LISTE (liste; apparence{; icône{; hauteurLigne{; doubleClic{; multiSélection{; modifiable}}}}})
515 Lire proprietes ressource (resType; resNum{; resFichier}) → Numérique
685 LIRE PROPRIETES SAISIE CHAMP (ptrChp|numTable{; numChamp}; nomEnum; obligatoire; nonSaisissable; nonModifiable)
687 LIRE PROPRIETES TABLE (ptrTable|numTable; invisible{; trigSvgdeNouv{; trigSvgdeEnr{; trigSupprEnr{; trigChargEnr}}}})
611 LIRE PROPRIETES UTILISATEUR (réfUtilisateur; nom; démarrage; motDePasse; nbUtilisations; dernièreUtilisation{; adhésions{; groupePropriétaire}})
663 LIRE RECT OBJET ({*; }objet; gauche; haut; droite; bas)
979 Lire reference barre menu {(process)} → RefMenu
508 LIRE RESSOURCE (resType; resNum; resDonnées{; resFichier})
506 Lire ressource chaine (resNum{; resFichier}) → Alpha
517 LIRE RESSOURCE ICONE (resNum; resDonnées{; resFichier})
502 LIRE RESSOURCE IMAGE (resNum; resDonnées{; resFichier})
504 Lire ressource texte (resNum{; resFichier}) → Texte
779 LIRE RESULTAT WEB SERVICE (valeurRetour{; nomRetour{; *}})
990 Lire source donnees courante → Chaîne
426 Lire style ligne menu (menu; ligneMenu{; process}) → Numérique
1014 LIRE TABLE SOURCE LISTBOX ({*; }objet; numTable{; tempo})
832 LIRE TABLEAUX LISTBOX ({*; }objet; tabNomsCols; tabNomsEntêtes; tabVarCols; tabVarEntêtes; tabVisibles; tabStyles)
708 Lire taquet impression (numTaquet) → Numérique
524 Lire texte dans conteneur → Chaîne
655 Lire texte edite → Texte
422 Lire texte ligne menu (menu; ligneMenu{; process}) → Alpha
430 Lire titre menu (menu{; process}) → Alpha
804 LIRE TITRES CHAMPS (table; titresChamps; numChamps)
803 LIRE TITRES TABLES (titresTables; numTables)
424 Lire touche ligne menu (menu; ligneMenu{; process}) → Numérique
991 Lire traduction chaine (resName) → Chaîne
958 LIRE TYPE DONNEES DANS CONTENEUR (signatures4D; typesNatifs{; nomsFormats})
826 Lire utilisateur par defaut → Numérique
371 LIRE VARIABLE PROCESS (process; varSource; varDestination{; varSource2; varDestination2; ...; varSourceN; varDestinationN})
74 LIRE VARIABLES (doc; variable{; variable2; ...; variableN})
683 LIRE VARIABLES FORMULAIRE WEB (tabNoms; tabValeurs)
703 LIRE ZONE IMPRESSION (hauteur{; largeur})
992 LISTE CODECS IMAGES (tabCodecs{; tabNoms})
1001 LISTE COMPOSANTS (tabComposants)
511 LISTE DE CHAINES VERS TABLEAU (resNum; chaînes{; resFichier})

474 LISTE DES DOCUMENTS (cheminAccès; documents)
473 LISTE DES DOSSIERS (cheminAccès; dossiers)
460 LISTE DES POLICES (polices)
471 LISTE DES VOLUMES (volumes)
957 LISTE ENUMERATIONS (tabNums; tabNoms)
621 Liste existante (liste) → Booléen
442 LISTE FENETRES (fenêtres{; *})
809 LISTE FORMULAIRES UTILISATEURS (table; formulaire; tabFormUtilisateurs)
564 LISTE IMAGES DANS BIBLIOTHEQUE (RefsImages; NomsImages)
789 LISTE IMPRIMANTES (tabNoms{; tabNomsAlt{; tabModèles}})
847 LISTE PLUGINS (tabNuméros; tabNoms)
500 LISTE RESSOURCES (resType; resNums; resNoms{; resFichier})
527 LISTE SEGMENTS DE DONNEES (segments)
989 LISTE SOURCES DONNEES (typeSource; tabNomsSources; tabPilotes)
681 LISTE TYPES IMAGES (tabFormats{; tabNoms})
499 LISTE TYPES RESSOURCE (resTypes{; resFichier})
556 LISTE VERS BLOB (liste; blob{; *})
22 Log (nombre) → Numérique
16 Longueur (chaîne) → Numérique

M

519 Mac vers ISO (texte) → Chaîne
463 Mac vers Windows (texte) → Chaîne
546 Macintosh commande enfoncee → Booléen
544 Macintosh control enfoncee → Booléen
545 Macintosh option enfoncee → Booléen
13 Majusc (laChaîne{; *}) → Alpha
543 Majuscule enfoncee → Booléen
656 MARQUER ENREGISTREMENTS ({table}{; nomEnsemble{; *}})
3 Max (séries) → Numérique
453 MAXIMISER FENETRE {(fenêtre)}
152 Menu choisi {(sousMenu)} → Numérique
88 MESSAGE (message)
704 Methode appelee sur erreur → Alpha
705 Methode appelee sur evenement → Alpha
4 Min (séries) → Numérique
454 MINIMISER FENETRE {(fenêtre)}
14 Minusc (laChaîne; *) → Alpha
492 Mode compile {(*)} → Booléen
32 Modifie (champ) → Booléen
57 MODIFIER ENREGISTREMENT ({table}{; }{*})
807 MODIFIER FORMULAIRE (table; formulaire{; formUtilisateur{; bibliothèque}})
204 MODIFIER SELECTION ({table}{; modeSélection{; saisieListe{; *{; *}}}})
203 MODIFIER SOUS ENREGISTREMENT (sousTable; formulaire{; *})

98 Modulo (nombre1; nombre2) → Numérique
24 Mois de (laDate) → Numérique
841 MONTRER GRILLE LISTBOX ({*; }objet; horizontal; vertical)
325 MONTRER PROCESS (process)
922 MONTRER SUR DISQUE (cheminAccès{; *})
2 Moyenne (séries) → Numérique

N

270 NE PAS VALIDER
315 Nil (unPointeur) → Booléen
101 Niveau → Numérique
961 Niveau de la transaction → Entier long
398 Niveau du trigger → Numérique
302 NIVEAUX DE RUPTURES (niveau{; sautPage})
538 Nom commande (commande) → Alpha
483 Nom de la machine → Alpha
256 Nom de la table (numTable | ptrTable) → Alpha
462 Nom de police (numéro) → Alpha
257 Nom du champ (champPtr | tableNum {; champNum}) → Alpha
484 Nom du possesseur → Alpha
684 Nom methode courante → Alpha
405 Nombre de lignes de menu (menu{; process}) → Numérique
404 Nombre de menus {(process)} → Numérique
459 Nombre de millisecondes → Entier long
259 Nombre de parametres → Numérique
335 Nombre de process → Entier
343 Nombre de process utilisateurs → Entier
458 Nombre de ticks → Numérique
437 Nombre ecrans → Entier long
380 Nombre elements ({*; }liste{; *}) → Entier long
342 Nombre utilisateurs → Entier
116 NOMMER ENSEMBLE ({table; }ensemble)
34 Non (booléen) → Booléen
1052 NOTIFIER MODIFICATION DOSSIER RESOURCES
926 Nouveau fichier historique → Texte
317 Nouveau process (méthode; pile{; nom{; param{; param2; ...; paramN}{; *}}}) → Numérique
668 Nouvel enregistrement {(laTable)} → Booléen
375 Nouvelle liste → RéfListe
11 Num (expression{; séparateur}) → Numérique
844 NUMERO COLONNE LISTBOX DEPLACEE ({*; }objet; ancPosition; nouvPosition)
246 Numero dans selection {(table)} → Numérique
897 Numero de ligne affichee → Entier long
461 Numero de police (nomPolice) → Entier long

114 Numero du jour (laDate) → Numérique
322 Numero du process courant → Numérique
243 Numero enregistrement {(laTable)} → Numérique
837 NUMERO LIGNE LISTBOX DEPLACEE ({*; }objet; ancPosition; nouvPosition)
244 Numerotation automatique {(table)} → Numérique

O

278 Objet focus → Pointeur
1018 OUVRIR CENTRE DE SECURITE
264 Ouvrir document (document{; leType{; mode}}) → DocRef
1047 OUVRIR FENETRE ADMINISTRATION
312 OUVRIR FICHIER DONNEES (cheminAccès)
497 Ouvrir fichier ressources (resNomFichier{; typeFichier}) → DocRef
903 OUVRIR PREFERENCES 4D (sélecteur)
995 OUVRIR TACHE IMPRESSION
673 OUVRIR URL WEB (url{; *})

P

276 Page formulaire courante → Numérique
275 Page impression → Numérique
249 PAGE PRECEDENTE
248 PAGE SUIVANTE
298 PARAMETRES DU GRAPHE (zoneGraphe; xmin; xmax; ymin; ymax; xprop; grilleX; grilleY; titre{; titre2; ...; titreN})
106 PARAMETRES IMPRESSION {(typeDial)}
993 PAS DE TABLE PAR DEFAUT
158 PAS DE TRACE
326 PASSER AU PREMIER PLAN (process)
30 Pendant → Booléen
304 Pointeur vers (nomVar) → Pointeur
542 Pop up menu (contenu{; parDéfaut{; coordX{; coordY}}}) → Numérique
1006 Pop up menu dynamique (menu{; parDéfaut{; coordX{; coordY}}}) → RefLigne
15 Position (àChercher; laChaîne{; départ{; longTrouvée{; *}}}) → Numérique
481 Position dans document (document) → Numérique
608 Position deposer {(numColonne)} → Numérique
629 Position element liste ({*; }liste; réfElément) → Numérique
161 POSITION MESSAGE (x; y)
468 POSITION SOURIS (sourisX; sourisY; boutonSouris{; *})
250 PREMIERE PAGE
446 Process de la fenetre {(fenêtre)} → Numérique
327 Process de premier plan {(*)} → Entier
672 Process interrompu → Booléen
439 PROFONDEUR ECRAN (profondeur; couleurs{; écran})

477 PROPRIETES DOCUMENT (document; verrouillé; invisible; créé le; créé à ; modifié le; modifié à)
399 PROPRIETES DU TRIGGER (niveauTrigger; evenementBase; tableNum; enregNum)
472 PROPRIETES DU VOLUME (volume; taille; utilisé; libre)
607 PROPRIETES GLISSER DEPOSER (srcObjet; srcElément; srcProcess)
457 PROPRIETES IMAGE (image; largeur; hauteur{; hOffset{; vOffset{; mode}}})
365 PROPRIETES PLATE FORME (plateForme{; système{; processeur{; langue}}})

Q

790 QR APPELER SUR COMMANDE (zone; nomMéthode)
771 QR BLOB VERS ETAT (zone; blob)
776 QR Chercher colonne (zone; expression) → Entier long
735 QR Creer zone hors ecran → Entier long
197 QR ETAT ({table; }document{; hiérarchique{; assistant{; recherche{; *}}}})
770 QR ETAT VERS BLOB (zone; blob)
746 QR EXECUTER (zone)
791 QR EXECUTER COMMANDE (zone; numCommande)
745 QR FIXER DESTINATION (zone; type{; spécificités})
767 QR FIXER DONNEES TOTAUX (zone; numColonne; numRupture; opérateur | valeur)
797 QR FIXER ENCADREMENTS (zone; colonne; ligne; encadrements; épaisseur{; couleur})
774 QR FIXER ENTETE ET PIED DE PAGE (zone; sélecteur; titreGauche; titreCentre; titreDroite; hauteur{; image{; alignementImage}})
761 QR FIXER ESPACEMENT TOTAUX (zone; sousTotal; valeur)
765 QR FIXER INFO COLONNE (zone; numColonne; titre; objet; cachée; taille; valeursRépétées; format)
763 QR FIXER INFO LIGNE (zone; ligne; cachée)
750 QR FIXER MODELE HTML (zone; modèle)
772 QR FIXER PROPRIETE DOCUMENT (zone; propriété; valeur)
759 QR FIXER PROPRIETE TEXTE (zone; numColonne; numLigne; propriété; valeur)
796 QR FIXER PROPRIETE ZONE (zone; propriété; valeur)
794 QR FIXER SELECTION (zone; gauche; haut; droite; bas)
757 QR FIXER TABLE ETAT (zone; table)
752 QR FIXER TRIS (zone; tabColonnes{; tabTris})
738 QR FIXER TYPE ETAT (zone; type)
748 QR INSERER COLONNE (zone; numColonne; objet)
747 QR Lire colonne deposee (zone) → Entier long
756 QR LIRE DESTINATION (zone; type{; spécificités})
768 QR LIRE DONNEES TOTAUX (zone; numColonne; numLigne; opérateur; texte)
798 QR LIRE ENCADREMENTS (zone; colonne; ligne; encadrement; épaisseur{; couleur})
775 QR LIRE ENTETE ET PIED DE PAGE (zone; sélecteur; titreGauche; titreCentre; titreDroit; hauteur{; image{; alignementImage}})
762 QR LIRE ESPACEMENT TOTAUX (zone; sousTotal; valeur)
766 QR LIRE INFO COLONNE (zone; numCol; titre; objet; cachée; taille; valeursRépétées; format)

769 QR Lire info ligne (zone; ligne) → Entier long
751 QR Lire modele HTML (zone) → Texte
773 QR Lire propriete document (zone; propriété) → Entier long
760 QR Lire propriete texte (zone; numColonne; numLigne; propriété) → Entier long
795 QR Lire propriete zone (zone; propriété) → Entier long
793 QR LIRE SELECTION (zone; gauche; haut{; droite{; bas}})
792 QR Lire statut commande (zone; numCommande{; valeur}) → Entier long
758 QR Lire table etat (zone) → Entier long
753 QR LIRE TRIS (zone; tabColonnes; tabTris)
755 QR Lire type etat (zone) → Entier long
764 QR Nombre de colonnes (zone) → Entier long
749 QR SUPPRIMER COLONNE (zone; numColonne)
754 QR SUPPRIMER ZONE HORS ECRAN (zone)
291 QUITTER 4D {(délai)}

R

539 Racine carree (nombre) → Numérique
320 REACTIVER PROCESS (process)
172 RECEVOIR BUFFER (varRéception)
79 RECEVOIR ENREGISTREMENT {(table)}
104 RECEVOIR PAQUET ({docRef; }réceptVar; stopCar | nbCar)
81 RECEVOIR VARIABLE (variable)
174 REDESSINER (objet)
456 REDESSINER FENETRE {(fenêtre)}
382 REDESSINER LISTE (liste)
890 REDIMENSIONNER FENETRE FORMULAIRE (largeur; hauteur)
351 REDUIRE SELECTION ({laTable; }nombre)
552 REEL VERS BLOB (réel; blob; formatRéel{; offset | *})
38 REFUSER {(champ)}
77 REGLER SERIE (port | opération{; param | document})
234 Remplacer caracteres (source; nouveau; position) → Alpha
233 Remplacer chaine (source; obsolète; nouveau{; remplacements}{; *}) → Alpha
695 RESOUDRE ALIAS (cheminAlias; cheminCible)
394 RESOUDRE POINTEUR (pointeur; nomVar; numTable; numChamp)
918 RESTITUER
120 REUNION (ensemble1; ensemble2; résultat)

S

6 SAUT DE PAGE {(* | >)}
887 SAUVEGARDER
856 SAX AJOUTER CDATA XML (document; données)
852 SAX AJOUTER COMMENTAIRE XML (document; commentaire)
851 SAX AJOUTER DOCTYPE XML (document; docType)

857 SAX AJOUTER INSTRUCTION DE TRAITEMENT (document; instruction)
855 SAX AJOUTER VALEUR ELEMENT XML (document; données{; *})
858 SAX ECRIRE OPTIONS XML (document; encodage{; autonome{; indentation}})
854 SAX FERMER ELEMENT XML (document)
878 SAX LIRE CDATA XML (document; valeur)
874 SAX LIRE COMMENTAIRE XML (document; commentaire)
876 SAX LIRE ELEMENT XML (document; nom; préfixe; nomsAttributs; valeursAttributs)
879 SAX LIRE ENTITE XML (document; nom; valeur)
875 SAX LIRE INSTRUCTION DE TRAITEMENT XML (document; nom; valeur)
860 SAX Lire noeud XML (document) → Entier long
877 SAX LIRE VALEUR ELEMENT XML (document; valeur)
873 SAX LIRE VALEURS DOCUMENT XML (document; encodage; version; autonome)
853 SAX OUVRIR ELEMENT XML (document; balise{; nomAttribut{; valeurAttribut}}{; nomAttribut2; valeurAttribut2; ...; nomAttributN; valeurAttributN})
921 SAX OUVRIR ELEMENT XML TABLEAUX (document; balise{; tabNomsAttributs{; tabValeursAttributs}}{; tabNomsAttributs2; tabValeursAttributs2; ...; tabNomsAttributsN; tabValeursAttributsN})
350 SCAN INDEX (leChamp; nombre{; > ou <})
368 SELECTION LIMITEE VERS TABLEAU (début; fin; champ | table; tableau{; champ2 | table2; tableau2; ...; champN | tableN; tableauN})
340 SELECTION RETOUR (leChamp)
260 SELECTION VERS TABLEAU (champ | table; tableau{; champ2 | table2; tableau2; ...; champN | tableN; tableauN})
956 Selectionner couleur RVB ({coulDefaut}{; }{message}) → Entier long
905 Selectionner document (répertoire; typesFichiers; titre; options{; sélectionnés}) → Chaîne
670 Selectionner dossier ({message}{; }{répertoire}) → Alpha
381 SELECTIONNER ELEMENTS PAR POSITION ({*; }liste; positionElém{; tabPositions})
630 SELECTIONNER ELEMENTS PAR REFERENCE (liste; réfElément{; tabRéfs})
912 SELECTIONNER LIGNE LISTBOX ({*; }objet; positionLigne{; action})
210 SELECTIONNER TEXTE (zone; débutSél; finSél)
308 Self → Pointeur
143 Semaphore (sémaphore{; nbTicks}) → Booléen
392 SIECLE PAR DEFAUT (siècle{; anPivot})
17 Sin (nombre) → Numérique
1 Somme (séries) → Numérique
28 Somme des carres (séries) → Numérique
12 Sous chaine (source; àPartirDe{; nbCars}) → Alpha
111 SOUS ENREGISTREMENT PRECEDENT (sousTable)
62 SOUS ENREGISTREMENT SUIVANT (sousTable)
7 Sous enregistrements trouves (sousTable) → Numérique
97 Sous total (valeurs{; sautPage}) → Numérique
824 SQL ANNULER CHARGEMENT
822 SQL CHARGER ENREGISTREMENT {(nombreEnr)}
820 SQL EXECUTER (instructionSQL{; objetLié}{; objetLié2; ...; objetLiéN})

881 SQL EXPORTER (tableSource{; projet{; *}})
821 SQL Fin de selection → Booléen
818 SQL FIXER OPTION (option; valeur)
823 SQL FIXER PARAMETRE (objet; typeParam)
880 SQL IMPORTER (tableSource{; projet{; *}})
825 SQL LIRE DERNIERE ERREUR (errCode; errTexte; errODBC; errSQLServer)
819 SQL LIRE OPTION (option; valeur)
817 SQL LOGIN{(nomSource; utilisateur; motDePasse{; *})}
872 SQL LOGOUT
658 STATISTIQUES DU CACHE WEB (pages; hits; usage)
330 Statut du process (process) → Numérique
53 STOCKER ENREGISTREMENT {(table)}
184 STOCKER ENSEMBLE (ensemble; document)
384 STOCKER LISTE (liste; nomListe)
43 STOCKER SUR LIEN (champ)
156 STOP
232 Supprimer chaine (source; position; nombreCar) → Alpha
830 SUPPRIMER COLONNE LISTBOX ({*; }objet; positionCol{; nombre})
560 SUPPRIMER DANS BLOB (blob; offset; nombre)
624 SUPPRIMER DANS LISTE ({*; }liste; réfElément | *{; *})
228 SUPPRIMER DANS TABLEAU (tableau; positionDépart{; combien})
159 SUPPRIMER DOCUMENT (document)
693 SUPPRIMER DOSSIER (dossier)
58 SUPPRIMER ENREGISTREMENT {(table)}
810 SUPPRIMER FORMULAIRE UTILISATEUR (table; formulaire; formUtilisateur)
567 SUPPRIMER IMAGE DANS BIBLIOTHEQUE (refImage | nomImage)
967 SUPPRIMER INDEX (ptrChp | nomIndex{; *})
914 SUPPRIMER LIGNE LISTBOX ({*; }objet; positionLigne)
413 SUPPRIMER LIGNE MENU (menu; ligneMenu{; process})
377 SUPPRIMER LISTE (liste{; *})
175 SUPPRIMER MESSAGES
501 SUPPRIMER RESSOURCE (resType; resNum{; resFichier})
66 SUPPRIMER SELECTION {(table)}
96 SUPPRIMER SOUS ENREGISTREMENT (sousTable)
615 SUPPRIMER UTILISATEUR (réfUtilisateur)
319 SUSPENDRE PROCESS (process)
1054 SVG Chercher ID element par coordonnees ({*;} objetImage; x; y) → Chaîne
1017 SVG EXPORTER VERS IMAGE (refElément; vVarImage{; typeExport})

T

252 Table (numTable | unPtr) → Num | Pointeur
627 Table du formulaire courant → Pointeur
46 TABLE PAR DEFAUT (laTable)
363 Table par defaut courante → Pointeur

218 TABLEAU ALPHA (longueurChaîne; nomTableau; taille{; taille2})
223 TABLEAU BOOLEEN (nomTableau; taille{; taille2})
646 TABLEAU BOOLEEN SUR ENSEMBLE (tabBooléen{; ensemble})
224 TABLEAU DATE (nomTableau; taille{; taille2})
220 TABLEAU ENTIER (nomTableau; taille{; taille2})
221 TABLEAU ENTIER LONG (nomTableau; taille{; taille2})
647 TABLEAU ENTIER LONG SUR SELECTION (table; tabEnrg{; tempo})
279 TABLEAU IMAGE (nomTableau; taille{; taille2})
718 TABLEAU MULTI TRI (tableau{; sensDuTri}{; tableau2; sensDuTri2; ...; tableauN; sensDuTriN})
280 TABLEAU POINTEUR (nomTableau; taille{; taille2})
219 TABLEAU REEL (nomTableau; taille{; taille2})
222 TABLEAU TEXTE (nomTableau; taille{; taille2})
287 TABLEAU VERS ENUMERATION (tableau; énumération{; réfEléments})
512 TABLEAU VERS LISTE DE CHAINES (chaînes; resNum{; resFichier})
261 TABLEAU VERS SELECTION (tableau; champ{; tableau2; champ2; ...; tableauN; champN})
605 Taille BLOB (blob) → Entier long
479 Taille document (document{; *}) → Numérique
356 Taille image (image) → Numérique
717 TAILLE OBJET OPTIMALE ({*; }objet; largeurOpti; hauteurOpti{; largeurMax})
274 Taille tableau (tableau) → Numérique
19 Tan (nombre) → Numérique
476 Tester chemin acces (cheminAccès) → Numérique
400 Tester conteneur (typeDonnées) → Numérique
652 Tester semaphore (sémaphore) → Booléen
209 TEXTE SELECTIONNE (zone; débutSél; finSél)
554 TEXTE VERS BLOB (texte; blob{; formatTexte{; offset | *}})
194 TITRE BOUTON ({*; }objet; libellBouton)
450 Titre fenetre {(fenêtre)} → Chaîne
109 TOUS LES SOUS ENREGISTREMENTS (sousTable)
47 TOUT SELECTIONNER {(table)}
157 TRACE
816 TRAITER BALISES HTML (donnéesEntrée; donnéesSortie)
397 Transaction en cours → Booléen
988 TRANSFORMER IMAGE (image; opérateur{; param1{; param2{; param3{; param4}}}})
49 TRIER ({table}{; champ{; > ou <{; champ2; > ou <2; ...; champN; > ou <N}{; *}}})
916 TRIER COLONNES LISTBOX ({*; }objet; numCol; sensDuTri{; numCol2; sensDuTri2; ...; numColN; sensDuTriN})
391 TRIER LISTE (liste{; > ou <})
300 TRIER PAR FORMULE (table{; expression{; > ou <}}{; expression2; > ou <2; ...; expressionN; > ou <N})
107 TRIER SOUS ENREGISTREMENTS (sousTable; sousChamp{; direction}{; sousChamp2; direction2; ...; sousChampN; directionN})
229 TRIER TABLEAU (tableau{; tableau2; ...; tableauN}{; sensDuTri})

U

V

1023 WA ACTUALISER URL ({*;}objet)
1039 WA AGRANDIR TEXTE PAGE({*;}objet)
1024 WA ARRETER CHARGEMENT URL ({*;}objet)
1049 WA Creer menu historique URL ({*;}objet{; direction}) → RefMenu
1043 WA EXECUTER FONCTION JAVASCRIPT ({*;}objet; fonctionJS; résultat|*{; param1;...;paramN})
1029 WA Executer JavaScript ({*;}objet; codeJS) → Chaîne
1037 WA FIXER CONTENU PAGE({*;}objet; contenu; baseURL)
1032 WA FIXER FILTRES LIENS EXTERNES({*;}objet; tabFiltres; tabAutorisRefus)
1030 WA FIXER FILTRES URL({*;}objet; tabFiltres; tabAutorisRefus)
1041 WA FIXER PREFERENCE({*;}objet; sélecteur; valeur)
1038 WA Lire contenu page({*;}objet) → Chaîne
1035 WA Lire dernier URL filtre ({*;}objet) → Chaîne
1034 WA LIRE DERNIERE ERREUR URL ({*;}objet; url; description; codeErreur)
1033 WA LIRE FILTRES LIENS EXTERNES({*;}objet; tabFiltres; tabAutorisRefus)
1031 WA LIRE FILTRES URL({*;}objet; tabFiltres; tabAutorisRefus)
1048 WA LIRE HISTORIQUE URL({*;}objet; tabUrls{; direction{; tabTitres}})
1042 WA LIRE PREFERENCE({*;}objet; sélecteur; valeur)
1036 WA Lire titre page ({*;}objet) → Chaîne
1025 WA Lire URL courant ({*;}objet) → Chaîne
1020 WA OUVRIR URL({*;}objet; url)
1021 WA OUVRIR URL PRECEDENT({*;}objet)
1022 WA OUVRIR URL SUIVANT({*;}objet)
1040 WA REDUIRE TEXTE PAGE({*;}objet)
1026 WA URL precedent disponible ({*;}objet) → Booléen
1027 WA URL suivant disponible ({*;}objet) → Booléen
563 Windows Alt enfoncee → Booléen
562 Windows Ctrl enfoncee → Booléen
464 Windows vers Mac (texte) → Chaîne

Constantes

Alignement objet

Commande(s) associée(s) : FIXER ALIGNEMENT, Lire alignement.

Constante	Type	Valeur
Aligné à droite	Entier long	4
Aligné à gauche	Entier long	2
Aligné par défaut	Entier long	1
Centré	Entier long	3

BLOB

Commande(s) associée(s) : BLOB vers entier, BLOB vers entier long, BLOB vers reel, BLOB vers texte, ENTIER LONG VERS BLOB, ENTIER VERS BLOB, LIRE PROPRIETES BLOB, REEL VERS BLOB, TEXTE VERS BLOB.

Constante	Type	Valeur
Format réel double Macintosh	Entier long	2
Format réel double PC	Entier long	3
Format réel étendu	Entier long	1
Format réel natif	Entier long	0
Mac Chaîne en C	Entier long	0
Mac Chaîne pascal	Entier long	1
Mac Texte avec longueur	Entier long	2
Mac Texte sans longueur	Entier long	3
Méthode de compression compacte	Entier long	1
Méthode de compression rapide	Entier long	2
Non compressé	Entier long	0
Ordre octets Macintosh	Entier long	1
Ordre octets natif	Entier long	0
Ordre octets PC	Entier long	2
UTF8 Chaîne en C	Entier long	4
UTF8 Texte avec longueur	Entier long	5
UTF8 Texte sans longueur	Entier long	6

Caractères latins ISO

Constante	Type	Valeur
ISO L1 0 majus circonflexe	Chaîne	Ô
ISO L1 a aigu	Chaîne	á
ISO L1 a circonflexe	Chaîne	â
ISO L1 a grave	Chaîne	à
ISO L1 A majus aigu	Chaîne	Á
ISO L1 A majus circonflexe	Chaîne	Â
ISO L1 A majus grave	Chaîne	À
ISO L1 A majus tilde	Chaîne	Ã
ISO L1 A majus umlaut	Chaîne	Ä
ISO L1 A majus umlaut	Chaîne	Å
ISO L1 a rond	Chaîne	å
ISO L1 a tilde	Chaîne	ã
ISO L1 a umlaut	Chaîne	ä
ISO L1 ae ligature	Chaîne	æ
ISO L1 AE majus ligature	Chaîne	&AELig;
ISO L1 c cédille	Chaîne	ç
ISO L1 C majus cédille	Chaîne	Ç
ISO L1 Copyright	Chaîne	©
ISO L1 e aigu	Chaîne	é
ISO L1 e circonflexe	Chaîne	ê
ISO L1 e grave	Chaîne	è
ISO L1 E majus aigu	Chaîne	É
ISO L1 E majus circonflexe	Chaîne	Ê
ISO L1 E majus grave	Chaîne	È
ISO L1 E majus unlaut	Chaîne	Ë
ISO L1 e umlaut	Chaîne	ë
ISO L1 Es zett allemand	Chaîne	ß
ISO L1 Et commercial	Chaîne	&
ISO L1 eth islandais	Chaîne	ð
ISO L1 Eth majus islandais	Chaîne	Ð
ISO L1 Guillemets	Chaîne	"
ISO L1 i aigu	Chaîne	í
ISO L1 i circonflexe	Chaîne	î
ISO L1 i grave	Chaîne	ì
ISO L1 I majus aigu	Chaîne	Í
ISO L1 I majus circonflex	Chaîne	Î
ISO L1 I majus grave	Chaîne	Ì
ISO L1 I majus umlaut	Chaîne	Ï
ISO L1 i umlaut	Chaîne	ï

Caractères latins ISO (suite)

Constante	Type	Valeur
ISO L1 Inférieur	Chaîne	&lt;
ISO L1 Marque déposée	Chaîne	&reg;
ISO L1 N majus tilde	Chaîne	&Ntilde;
ISO L1 n tilde	Chaîne	&ntilde;
ISO L1 o aigu	Chaîne	&oacute;
ISO L1 o barré	Chaîne	&oslash;
ISO L1 o circonflexe	Chaîne	&ocirc;
ISO L1 o grave	Chaîne	&ograve;
ISO L1 O majus aigu	Chaîne	&Oacute;
ISO L1 O majus barré	Chaîne	&Oslash;
ISO L1 O majus grave	Chaîne	&Ograve;
ISO L1 O majus tilde	Chaîne	&Otilde;
ISO L1 O majus umlaut	Chaîne	&Ouml;
ISO L1 o tilde	Chaîne	&otilde;
ISO L1 o umlaut	Chaîne	&ouml;
ISO L1 Supérieur	Chaîne	&gt;
ISO L1 thorn islandais	Chaîne	&thorn;
ISO L1 THORN majus islandais	Chaîne	&THORN;
ISO L1 u aigu	Chaîne	&uacute;
ISO L1 u circonflexe	Chaîne	&ucirc;
ISO L1 u grave	Chaîne	&ugrave;
ISO L1 U majus aigu	Chaîne	&Uacute;
ISO L1 U majus circonflexe	Chaîne	&Ucirc;
ISO L1 U majus grave	Chaîne	&Ugrave;
ISO L1 U majus umlaut	Chaîne	&Uuml;
ISO L1 u umlaut	Chaîne	&uuml;
ISO L1 y aigu	Chaîne	&yacute;
ISO L1 Y majus aigu	Chaîne	&Yacute;
ISO L1 y umlaut	Chaîne	&yuml;

Chercher fenetre

Commande(s) associée(s) : Chercher fenetre.

Constante	Type	Valeur
Dans barre de menu	Entier long	1
Dans barre de titre	Entier long	4
Dans case de contrôle de taille	Entier long	5
Dans case de fermeture	Entier long	6
Dans case de zoom	Entier long	8
Dans fenêtre système	Entier long	2
Dans zone contenu	Entier long	3

Codes ASCII

Commande(s) associée(s) : APPELER SUR EVENEMENT, Caractere.

Constante	Type	Valeur
Apostrophe	Entier long	39
Arobase	Entier long	64
ASCII ACK	Entier long	6
ASCII BEL	Entier long	7
ASCII BS	Entier long	8
ASCII CAN	Entier long	24
ASCII CR	Entier long	13
ASCII DC1	Entier long	17
ASCII DC2	Entier long	18
ASCII DC3	Entier long	19
ASCII DC4	Entier long	20
ASCII DEL	Entier long	127
ASCII DLE	Entier long	16
ASCII EM	Entier long	25
ASCII ENQ	Entier long	5
ASCII EOT	Entier long	4
ASCII ESC	Entier long	27
ASCII ETB	Entier long	23
ASCII ETX	Entier long	3
ASCII FF	Entier long	12
ASCII FS	Entier long	28
ASCII GS	Entier long	29
ASCII HT	Entier long	9
ASCII LF	Entier long	10
ASCII NAK	Entier long	21
ASCII NUL	Entier long	0
ASCII RS	Entier long	30
ASCII SI	Entier long	15
ASCII SO	Entier long	14
ASCII SOH	Entier long	1
ASCII SP	Entier long	32
ASCII STX	Entier long	2
ASCII SUB	Entier long	26
ASCII SYN	Entier long	22
ASCII US	Entier long	31
ASCII VT	Entier long	11

Codes ASCII (suite)

Constante	Type	Valeur
Echappement	Entier long	27
Effacement arrière	Entier long	8
Entrée	Entier long	3
Espace insécable	Entier long	202
Espacement	Entier long	32
Guillemets	Entier long	34
Point	Entier long	46
Retour à la ligne	Entier long	10
Retour chariot	Entier long	13
Tabulation	Entier long	9

Communications

Commande(s) associée(s) : REGLER SERIE.

Constante	Type	Valeur
Bit de stop deux	Entier long	-16384
Bit de stop un	Entier long	16384
Bit de stop un et demi	Entier long	-32768
Bits de données 5	Entier long	0
Bits de données 6	Entier long	2048
Bits de données 7	Entier long	1024
Bits de données 8	Entier long	3072
Parité impaire	Entier long	4096
Parité paire	Entier long	12288
Pas de parité	Entier long	0
Port imprimante MacOS	Entier long	0
Port série MacOS	Entier long	1
Protocole aucun	Entier long	0
Protocole DTR	Entier long	30
Protocole XONXOFF	Entier long	20
Vitesse 115200	Entier long	1022
Vitesse 1200	Entier long	94
Vitesse 1800	Entier long	62
Vitesse 19200	Entier long	4
Vitesse 230400	Entier long	1021
Vitesse 2400	Entier long	46
Vitesse 300	Entier long	380
Vitesse 3600	Entier long	30
Vitesse 4800	Entier long	22
Vitesse 57600	Entier long	0
Vitesse 600	Entier long	189
Vitesse 7200	Entier long	14
Vitesse 9600	Entier long	10

Compression images

Commande(s) associée(s) : CHARGER ET COMPRESSER IMAGE, COMPRESSER FICHIER IMAGE, COMPRESSER IMAGE.

Constante	Type	Valeur
QT Compresseur animation	Chaîne	rle
QT Compresseur Compact Vidéo	Chaîne	cdvc
QT Compresseur graphiques	Chaîne	smc
QT Compresseur photo	Chaîne	jpeg
QT Compresseur RAW	Chaîne	raw
QT Compresseur vidéo	Chaîne	rpza

Conteneur de données

Commande(s) associée(s) : AJOUTER DONNEES AU CONTENEUR, LIRE DONNEES CONTENEUR, Tester conteneur.

Constante	Type	Valeur
Données absentes conteneur	Entier long	-102
Données image	Chaîne	PICT
Données texte	Chaîne	TEXT

Couleurs

Commande(s) associée(s) : CHOIX COULEUR.

Constante	Type	Valeur
Blanc	Entier long	0
Bleu	Entier long	6
Bleu clair	Entier long	7
Bleu foncé	Entier long	5
Gris	Entier long	14
Gris clair	Entier long	12
Gris foncé	Entier long	11
Jaune	Entier long	1
Marron	Entier long	13
Marron foncé	Entier long	10
Noir	Entier long	15
Orange	Entier long	2
Rouge	Entier long	3
Vert	Entier long	8
Vert foncé	Entier long	9
Violet	Entier long	4

Creer fenetre

Commande(s) associée(s) : Creer fenetre, Creer fenetre externe.

Constante	Type	Valeur
Aspect métal	Entier long	2048
Avec barre de titre active	Entier long	1
Avec case de contrôle de taille	Entier long	4
Avec case de zoom	Entier long	8
Avec titre de fenêtre	Entier long	2
Dialogue modal	Entier long	1
Dialogue modal déplaçable	Entier long	5
Dialogue ombré	Entier long	3
Dialogue simple	Entier long	2
Fenêtre à coins arrondis	Entier long	16
Fenêtre feuille	Entier long	33
Fenêtre feuille redim	Entier long	34
Fenêtre palette	Entier long	1984
Fenêtre pop up	Entier long	32
Fenêtre standard	Entier long	8
Fenêtre standard de taille fixe	Entier long	4
Fenêtre standard sans zoom	Entier long	0
Mode compositing		4096

Creer fenetre formulaire

Commande(s) associée(s) : Creer fenetre formulaire.

Constante	Type	Valeur
A droite	Entier long	196608
A gauche	Entier long	131072
Centrée horizontalement	Entier long	65536
Centrée verticalement	Entier long	262144
En bas	Entier long	393216
En haut	Entier long	327680
Form dialogue modal	Entier long	1
Form dialogue modal déplaçable	Entier long	5
Form fenêtre feuille	Entier long	33
Form fenêtre palette	Entier long	1984
Form fenêtre pop up	Entier long	32
Form fenêtre standard	Entier long	8
Form Mode compositing	Entier long	4096

Dictionnaires

Commande(s) associée(s) : CHANGER DICTIONNAIRE.

Constante	Type	Valeur
Dictionnaire allemand	Entier long	131584
Dictionnaire anglais	Entier long	69632
Dictionnaire espagnol	Entier long	196608
Dictionnaire français	Entier long	262144
Dictionnaire norvégien	Entier long	589824

Documents système

Commande(s) associée(s) : Ouvrir document, Selectionner document, Tester chemin acces.

Constante	Type	Valeur
Est un document	Entier long	1
Est un répertoire	Entier long	0
Fichiers multiples	Entier long	1
Lecture et écriture	Entier long	0
Lire chemin accès	Entier long	3
Mode écriture	Entier long	1
Mode lecture	Entier long	2
Ouverture progiciel	Entier long	2
Sélection alias	Entier long	8
Sélection progiciel	Entier long	4
Utiliser fenêtre feuille	Entier long	16

Dossier Système

Commande(s) associée(s) : Dossier systeme.

Constante	Type	Valeur
Applications ou Program Files	Entier long	16
Bureau	Entier long	15
Démarrage Win	Entier long	5
Démarrage Win_Tous	Entier long	4
Favoris Win	Entier long	14
Menu Démarrer Win	Entier long	9
Menu Démarrer Win_Tous	Entier long	8
Polices	Entier long	1
Préférences utilisateur	Entier long	3
Préférences utilisateur_Tous	Entier long	2
System Win	Entier long	12
System32 Win	Entier long	13
Système	Entier long	0
_O_Extensions Mac	Entier long	10
_O_Ouverture extinction Mac	Entier long	7
_O_Ouverture extinction Mac_Tou	Entier long	6
_O_Tableaux de bord Mac	Entier long	11

Environnement 4D

Commande(s) associée(s) : Dossier 4D, FIXER PARAMETRE MACRO, LIRE PARAMETRE MACRO, Type application, Type version.

Constante	Type	Valeur
4D First	Entier long	6
4D mode distant	Entier long	4
4D mode local	Entier long	0
4D Server	Entier long	5
4D Volume Desktop	Entier long	1
Dossier 4D actif	Entier long	0
Dossier base	Entier long	4
Dossier base 4D Client	Entier long	3
Dossier base syntaxe Unix	Entier long	5
Dossier Extras	Entier long	2
Dossier Licenses	Entier long	1
Dossier Logs	Entier long	7
Dossier racine HTML	Entier long	8
Dossier Resources courant	Entier long	6
Texte méthode	Entier long	1
Texte méthode surligné	Entier long	2
Version de démonstration	Entier long	1
Version standard	Entier long	0
_O_4D Desktop	Entier long	3
_O_4D Interpreted Desktop	Entier long	2

Euro monnaies

Commande(s) associée(s) : Convertisseur Euro.

Constante	Type	Valeur
Drachme grecque	Chaîne	GRD
Escudo portugais	Chaîne	PTE
Euro	Chaîne	EUR
Florin néerlandais	Chaîne	NLG
Franc belge	Chaîne	BEF
Franc français	Chaîne	FRF
Franc luxembourgeois	Chaîne	LUF
Lire italienne	Chaîne	ITL
Livre irlandaise	Chaîne	IEP
Mark allemand	Chaîne	DEM
Mark finlandais	Chaîne	FIM
Peseta espagnole	Chaîne	ESP
Schilling autrichien	Chaîne	ATS

Evénements (Modifiers)

Commande(s) associée(s) : FIXER RACCOURCI LIGNE MENU, GENERER EVENEMENT, GENERER FRAPPE CLAVIER, Lire modificateurs ligne menu.

Constante	Type	Valeur
Bit activation fenêtre	Entier long	0
Bit bouton souris	Entier long	7
Bit touche commande	Entier long	8
Bit touche contrôle	Entier long	12
Bit touche contrôle droite	Entier long	15
Bit touche majuscule	Entier long	9
Bit touche majuscule droite	Entier long	13
Bit touche option	Entier long	11
Bit touche option droite	Entier long	14
Bit touche verrouillage maj	Entier long	10
Masque activation fenêtre	Entier long	1
Masque bouton souris	Entier long	128
Masque touche commande	Entier long	256
Masque touche contrôle	Entier long	4096
Masque touche contrôle droite	Entier long	32768
Masque touche majuscule	Entier long	512
Masque touche majuscule droite	Entier long	8192
Masque touche option	Entier long	2048
Masque touche option droite	Entier long	16384
Masque touche verrouillage maj	Entier long	1024

Evénements (types)

Commande(s) associée(s) : GENERER EVENEMENT.

Constante	Type	Valeur
Activation fenêtre	Entier long	8
Bouton souris enfoncé	Entier long	1
Bouton souris relâché	Entier long	2
Evénement disque	Entier long	7
Evénement nul	Entier long	0
Evénement système	Entier long	15
Mise à jour fenêtre	Entier long	6
Répétition touche	Entier long	5
Touche enfoncée	Entier long	3
Touche relâchée	Entier long	4

Evénements formulaire

Commande(s) associée(s) : Evenement formulaire.

Constante	Type	Valeur
Sur activation	Entier long	11
Sur affichage corps	Entier long	8
Sur appel extérieur	Entier long	10
Sur appel zone du plug in	Entier long	19
Sur après frappe clavier	Entier long	28
Sur après modification	Entier long	45
Sur après tri	Entier long	30
Sur avant frappe clavier	Entier long	17
Sur avant saisie	Entier long	41
Sur case de fermeture	Entier long	22
Sur chargement	Entier long	1
Sur chargement ligne	Entier long	40
Sur chargement ressource URL	Entier long	48
Sur clic	Entier long	4
Sur clic entête	Entier long	42
Sur clic flèche	Entier long	38
Sur clic long	Entier long	39
Sur contracter	Entier long	44
Sur début chargement URL	Entier long	47
Sur début glisser	Entier long	46
Sur début survol	Entier long	35
Sur déplacement colonne	Entier long	32
Sur déplacement ligne	Entier long	34
Sur déployer	Entier long	43
Sur déposer	Entier long	16
Sur désactivation	Entier long	12
Sur données modifiées	Entier long	20
Sur double clic	Entier long	13
Sur entête	Entier long	5
Sur erreur chargement URL	Entier long	50
Sur fermeture corps	Entier long	26
Sur filtrage URL	Entier long	51
Sur fin chargement URL	Entier long	49
Sur fin survol	Entier long	36
Sur gain focus	Entier long	15
Sur glisser	Entier long	21

Evénements formulaire (suite)

Constante	Type	Valeur
Sur impression corps	Entier long	23
Sur impression pied de page	Entier long	7
Sur impressions sous total	Entier long	6
Sur libération	Entier long	24
Sur menu sélectionné	Entier long	18
Sur minuteur	Entier long	27
Sur nouvelle sélection	Entier long	31
Sur ouverture corps	Entier long	25
Sur ouverture lien externe	Entier long	52
Sur perte focus	Entier long	14
Sur redimensionnement	Entier long	29
Sur redimensionnement colonne	Entier long	33
Sur refus ouverture fenêtre	Entier long	53
Sur survol	Entier long	37
Sur validation	Entier long	3

Evénements moteur

Commande(s) associée(s) : Evenement moteur, PROPRIETES DU TRIGGER.

Constante	Type	Valeur
Sur sauvegarde enregistrement	Entier long	2
Sur sauvegarde nouvel enreg	Entier long	1
Sur suppression enregistrement	Entier long	3
_O_Sur chargement enreg	Entier long	4

Expressions

Commande(s) associée(s) : RECEVOIR BUFFER, RECEVOIR PAQUET.

Constante	Type	Valeur
MAXENT	Entier long	32767
MAXLONG	Entier long	2147483647
MAXLONGTEXTEAVANTV11	Entier long	32000

FIXER COULEUR RVB

Commande(s) associée(s) : FIXER COULEURS RVB, Selectionner couleur RVB.

Constante	Type	Valeur
Coul arrière plan	Entier long	-2
Coul claire	Entier long	-4
Coul de fond texte sélect	Entier long	-7
Coul fond élément sélect désact	Entier long	-11
Coul fond ligne menu sélect	Entier long	-9
Coul premier plan	Entier long	-1
Coul sombre	Entier long	-3
Coul texte ligne menu sélect	Entier long	-10
Coul texte sélect	Entier long	-8

Fonctions mathématiques

Commande(s) associée(s) : Arctan, Cos, Sin, Tan.

Constante	Type	Valeur
Degré	Numérique	0.0174532925199432958
Nombre e	Numérique	2.71828182845904524
Pi	Numérique	3.141592653589793239
Radian	Numérique	57.2957795130823088

Formatages système

Commande(s) associée(s) : LIRE FORMATAGE SYSTEME.

Constante	Type	Valeur
Libellé AM heure système	Entier long	18
Libellé PM heure système	Entier long	19
Motif date abrégé	Entier long	7
Motif date court	Entier long	6
Motif date long	Entier long	8
Motif heure abrégé	Entier long	4
Motif heure court	Entier long	3
Motif heure long	Entier long	5
Position année date courte	Entier long	17
Position jour date courte	Entier long	15
Position mois date courte	Entier long	16
Séparateur date	Entier long	13
Séparateur de milliers	Entier long	1
Séparateur décimal	Entier long	0
Séparateur heure	Entier long	14
Symbole monétaire	Entier long	2

Formats d'affichage des dates

Commande(s) associée(s) : Chaine, CHOIX FORMATAGE, Lire formatage.

Constante	Type	Valeur
Interne date abrégé	Entier long	6
Interne date court	Entier long	7
Interne date court spécial	Entier long	4
Interne date long	Entier long	5
ISO Date	Entier long	8
Système date abrégé	Entier long	2
Système date court	Entier long	1
Système date long	Entier long	3
Vide si date nulle	Entier long	100

Formats d'affichage des heures

Commande(s) associée(s) : Chaine, CHOIX FORMATAGE, Lire formatage.

Constante	Type	Valeur
h mn	Entier long	2
h mn Matin Après Midi	Entier long	5
h mn s	Entier long	1
Heures Minutes	Entier long	4
Heures Minutes Secondes	Entier long	3
ISO Heure	Entier long	8
Minutes secondes	Entier long	7
mn s	Entier long	6
Système heure court	Entier long	9
Système heure long	Entier long	11
Système heure long abrégé	Entier long	10
Vide si heure nulle	Entier long	100

Formats d'affichage des images

Commande(s) associée(s) : CHOIX FORMATAGE, CREER IMAGETTE, Lire formatage.

Constante	Type	Valeur
Mosaïque	Entier long	7
Non tronquée	Entier long	2
Proportionnelle	Entier long	5
Proportionnelle centrée	Entier long	6
Sur fond	Entier long	3
Tronquée centrée	Entier long	1
Tronquée non centrée	Entier long	4

Interfaces de plate-forme

Commande(s) associée(s) : FIXER INTERFACE, Lire interface.

Constante	Type	Valeur
Mac OS 7	Entier long	0
Mac OS 9	Entier long	3
Plate forme automatique	Entier long	-1
Thème Mac	Entier long	4
Windows 3.11, NT 3.51	Entier long	1
Windows 9x	Entier long	2

Journal d'événements

Commande(s) associée(s) : ENREGISTRER EVENEMENT.

Constante	Type	Valeur
Message d’avertissement	Entier long	1
Message d’erreur	Entier long	2
Message d’information	Entier long	0
Vers historique commandes 4D	Entier long	3
Vers historique requêtes 4D	Entier long	2
Vers message débogage	Entier long	1
Vers Observateur Windows	Entier long	0

Jours et mois

Commande(s) associée(s) : Mois de, Numero du jour.

Constante	Type	Valeur
Août	Entier long	8
Avril	Entier long	4
Décembre	Entier long	12
Dimanche	Entier long	1
Février	Entier long	2
Janvier	Entier long	1
Jeudi	Entier long	5
Juillet	Entier long	7
Juin	Entier long	6
Lundi	Entier long	2
Mai	Entier long	5
Mardi	Entier long	3
Mars	Entier long	3
Mercredi	Entier long	4
Novembre	Entier long	11
Octobre	Entier long	10
Samedi	Entier long	7
Septembre	Entier long	9
Vendredi	Entier long	6

Licence disponible

Commande(s) associée(s) : ECRIRE ACCES PLUGIN, Licence disponible, Lire acces plugin, LISTE PLUGINS.

Constante	Type	Valeur
Licence 4D Draw	Entier long	808464694
Licence 4D for ADO	Entier long	808465714
Licence 4D for MySQL	Entier long	808465712
Licence 4D for OCI	Entier long	808465208
Licence 4D for PostgreSQL	Entier long	808465713
Licence 4D for Sybase	Entier long	808465715
Licence 4D ODBC Pro	Entier long	808464946
Licence 4D SOAP	Entier long	808465464
Licence 4D SOAP locale	Entier long	808531000
Licence 4D SOAP une connexion	Entier long	825242680
Licence 4D View	Entier long	808465207
Licence 4D Web	Entier long	808464945
Licence 4D Web locale	Entier long	808530481
Licence 4D Web une connexion	Entier long	825242161
Licence 4D Write	Entier long	808464697
Licence Serveur SQL 4D	Entier long	808464949
Licence Serveur SQL 4D 1 conn	Entier long	825242165
Licence Serveur SQL 4D locale		808530485
Licence SOAP 4D Client	Entier long	808465465
Licence Web 4D Client	Entier long	808465209

Liens

Commande(s) associée(s) : FIXER LIEN CHAMP, LIRE LIEN CHAMP.

Constante	Type	Valeur
Automatique	Entier long	3
Configuration structure	Entier long	1
Manuel	Entier long	2
Ne pas changer	Entier long	0
Pas de lien	Entier long	0

List box

Commande(s) associée(s) : Lire information listbox, SELECTIONNER LIGNE LISTBOX.

Constante	Type	Valeur
Affichage barre déf hor listbox	Entier long	2
Affichage barre déf ver listbox	Entier long	4
Affichage entête listbox	Entier long	0
Ajouter à sélection listbox	Entier long	1
Hauteur barre déf hor listbox	Entier long	3
Hauteur entête listbox	Entier long	1
Largeur barre déf ver listbox	Entier long	5
Position barre déf hor listbox	Entier long	6
Position barre déf ver listbox	Entier long	7
Remplacer sélection listbox	Entier long	0
Supprimer de sélection listbox	Entier long	2

Listes hiérarchiques

Commande(s) associée(s) : CHANGER PROPRIETES LISTE, FIXER PARAMETRE ELEMENT, LIRE PARAMETRE ELEMENT, LIRE PROPRIETES LISTE.

Constante	Type	Valeur
A la Macintosh	Entier long	1
A la Windows	Entier long	2
Réf icône Macintosh	Entier long	860
Réf icône Windows	Entier long	138
Texte supplémentaire	Chaîne	4D_additional_text
Utiliser réf image	Entier long	131072
Utiliser ressource PICT	Entier long	65536

Maintenance fichier de données

Commande(s) associée(s) : Compacter fichier donnees, VERIFIER FICHIER DONNEES.

Constante	Type	Valeur
Créer un process	Entier long	32768
Déplacer dans Replaced files	Entier long	4
Dialogue nouveau fichier	Entier long	1
Ne pas compacter les index	Entier long	2
Ne pas créer d'historique	Entier long	16384
Nouveau fichier	Entier long	0
Renuméroter les enregistrements	Entier long	1
Tout vérifier	Entier long	16
Utiliser fichier sélectionné	Entier long	2
Vérifier enregistrements	Entier long	4
Vérifier index	Entier long	8

Moteur de la base

Commande(s) associée(s) : Numero enregistrement.

Constante	Type	Valeur
Aucun enregistrement courant	Entier long	-1
Est un nouvel enregistrement	Entier long	-3

Numéros de port TCP

Commande(s) associée(s) : FIXER PARAMETRE BASE, Lire parametre base.

Constante	Type	Valeur
TCP Authentication	Entier long	113
TCP DNS	Entier long	53
TCP Finger	Entier long	79
TCP FTP Control	Entier long	21
TCP FTP Data	Entier long	20
TCP Gopher	Entier long	70
TCP HTTP WWW	Entier long	80
TCP IMAP3	Entier long	220
TCP Kerberos	Entier long	88
TCP KLogin	Entier long	543
TCP Nickname	Entier long	43
TCP NNTP	Entier long	119
TCP NTalk	Entier long	518
TCP NTP	Entier long	123
TCP PMCP	Entier long	1643
TCP PMD	Entier long	1642
TCP POP3	Entier long	110
TCP Printer	Entier long	515
TCP RADACCT	Entier long	1646
TCP RADIUS	Entier long	1645
TCP Remote Cmd	Entier long	514
TCP Remote Exec	Entier long	512
TCP Remote Login	Entier long	513
TCP Router	Entier long	520
TCP SMTP	Entier long	25
TCP SNMP	Entier long	161
TCP SNMPTRAP	Entier long	162
TCP SUN RPC	Entier long	111
TCP Talk	Entier long	517
TCP Telnet	Entier long	23
TCP TFTP	Entier long	69
TCP UUCP	Entier long	540
TCP UUCP RLOGIN	Entier long	541

Options d'impression

Commande(s) associée(s) : FIXER OPTION IMPRESSION, LIRE OPTION IMPRESSION, VALEURS OPTION IMPRESSION.

Constante	Type	Valeur
Option alimentation	Entier long	5
Option couleur	Entier long	8
Option destination	Entier long	9
Option échelle	Entier long	3
Option masquer progression impr	Entier long	14
Option mode impression Mac	Entier long	13
Option nom document à imprimer	Entier long	12
Option nombre copies	Entier long	4
Option orientation	Entier long	2
Option papier	Entier long	1
Option recto verso	Entier long	11

Paramètres de formulaire

Commande(s) associée(s) : LIRE PARAMETRE FORMULAIRE, MODIFIER SELECTION, VISUALISER SELECTION.

Constante	Type	Valeur
Objets non inversés	Entier long	0
Pas de sélection	Entier long	0
Sélection multiple	Entier long	2
Sélection unique	Entier long	1

Paramètres de la base

Commande(s) associée(s) : FIXER PARAMETRE BASE, Lire parametre base.

Constante	Type	Valeur
Adresse IP d'écoute	Entier long	16
Appels système 4D mode distant	Entier long	12
Appels système 4D mode local	Entier long	10
Appels système 4D Server	Entier long	11
Casse caractères moteur SQL	Entier long	44
Chercher par formule serveur	Entier long	46
Client Adresse IP d'écoute	Entier long	23
Client Enreg requêtes Web	Entier long	30
Client Jeu de caractères	Entier long	24
Client Maximum process Web	Entier long	20
Client Minimum process Web	Entier long	19
Client Numéro de port	Entier long	22
Client Numéro de port HTTPS	Entier long	40
Client Proc Web simultanés maxi	Entier long	25
Client Taille max requêtes Web	Entier long	21
Compression index	Entier long	4
Enreg événements debogage	Entier long	34
Enreg requêtes 4D Server	Entier long	28
Enreg requêtes client	Entier long	45
Enreg requêtes Web	Entier long	29
Inversion des objets	Entier long	37
Jeu de caractères	Entier long	17
Jointures CHERCHER PAR FORMULE	Entier long	49
Maximum process Web	Entier long	7
Minimum process Web	Entier long	6
Mode conversion Web	Entier long	8
Mode écriture cache	Entier long	26
Mode Unicode	Entier long	41
Niveau de compression HTTP	Entier long	50
Numéro automatique table	Entier long	31
Numéro de port HTTPS	Entier long	39
Numéro du port	Entier long	15
Numéro du port client serveur	Entier long	35
Optimisation accès séq	Entier long	2
Précision affichage réels	Entier long	32
Process Web simultanés maxi	Entier long	18

Paramètres de la base (suite)

Constante	Type	Valeur
Ratio chercher dans sélec séq	Entier long	5
Ratio de tri séq	Entier long	1
Ratio valeurs distinctes séq	Entier long	3
Seuil de compression HTTP	Entier long	51
Signature WEDD	Entier long	36
SQL Autocommit	Entier long	43
Synchro auto dossier Resources	Entier long	48
Taille cache données	Entier long	9
Taille maximum requêtes Web	Entier long	27
Taille mémoire temporaire	Entier long	42
TCP_NODELAY	Entier long	33
Timeout 4D mode distant	Entier long	14
Timeout 4D Server	Entier long	13
Timeout connexions inactives	Entier long	54
Trier par formule serveur	Entier long	47

PROFONDEUR ECRAN

Commande(s) associée(s) : FIXER PROFONDEUR ECRAN, PROFONDEUR ECRAN.

Constante	Type	Valeur
Deux cent cinquante six coul	Entier long	8
Est en couleurs	Entier long	1
Est en niveaux de gris	Entier long	0
Milliers de couleurs	Entier long	16
Millions de couleurs 24 bits	Entier long	24
Millions de couleurs 32 bits	Entier long	32
Noir et blanc	Entier long	0
Quatre couleurs	Entier long	2
Seize couleurs	Entier long	4

Propriétés des lignes de menu

Commande(s) associée(s) : FIXER PROPRIETE LIGNE MENU, LIRE PROPRIETE LIGNE MENU.

Constante	Type	Valeur
Action standard associée	Chaîne	4D_standard_action
Autorisations d'accèss	Chaîne	4D_access_group
Démarrer un process	Chaîne	4D_start_new_process

Propriétés des ressources

Commande(s) associée(s) : ECRIRE PROPRIETES RESSOURCE, Lire proprietes ressource.

Constante	Type	Valeur
Bit ressource heap système	Entier long	6
Bit ressource modifiée	Entier long	1
Bit ressource préchargée	Entier long	2
Bit ressource protégée	Entier long	3
Bit ressource purgeable	Entier long	5
Bit ressource verrouillée	Entier long	4
Masque ressource heap système	Entier long	64
Masque ressource modifiée	Entier long	2
Masque ressource préchargée	Entier long	4
Masque ressource protégée	Entier long	8
Masque ressource purgeable	Entier long	32
Masque ressource verrouillée	Entier long	16

Propriétés plate-forme

Les constantes préfixées _O_ sont obsolètes et ne peuvent plus être retournées par la commande. Elles sont conservées par compatibilité.

Commande(s) associée(s) : PROPRIETES PLATE FORME.

Constante	Type	Valeur
Compatible Intel	Entier long	586
Mac OS	Entier long	2
Power PC	Entier long	406
Windows	Entier long	3
_O_INTEL 386	Entier long	386
_O_INTEL 486	Entier long	486
_O_Macintosh 68K	Entier long	1
_O_PowerPC 601	Entier long	601
_O_PowerPC 603	Entier long	603
_O_PowerPC 604	Entier long	604
_O_PowerPC G3	Entier long	510

QR Commandes

Commande(s) associée(s) : QR EXECUTER COMMANDE, QR Lire statut commande.

Constante	Type	Valeur
qr cmd ajouter colonne	Entier long	2608
qr cmd aligné à droite	Entier long	505
qr cmd aligné à gauche	Entier long	503
qr cmd aligné par défaut	Entier long	512
qr cmd apercu	Entier long	2007
qr cmd cacher colonne	Entier long	2602
qr cmd cacher ligne	Entier long	2607
qr cmd centré	Entier long	504
qr cmd déplacer à droite	Entier long	3003
qr cmd déplacer à gauche	Entier long	3002
qr cmd destination 4D View	Entier long	2503
qr cmd destination fichier	Entier long	2501
qr cmd destination fichier HTML	Entier long	2504
qr cmd destination graphe	Entier long	2502
qr cmd destination imprimante	Entier long	2500
qr cmd écart type	Entier long	513
qr cmd éditer colonne	Entier long	2603
qr cmd encadrements	Entier long	2609
qr cmd enregistrer	Entier long	2002
qr cmd enregistrer sous	Entier long	2003
qr cmd entete et pied de page	Entier long	2005
qr cmd espacement des totaux	Entier long	2610
qr cmd executer	Entier long	2008
qr cmd format	Entier long	2606
qr cmd gras	Entier long	500
qr cmd insérer colonne	Entier long	2600
qr cmd italique	Entier long	501
qr cmd largeur automatique	Entier long	2605
qr cmd liste déroulante police	Entier long	1000
qr cmd max	Entier long	509
qr cmd min	Entier long	508
qr cmd mise en forme	Entier long	2611
qr cmd mise en page	Entier long	2006
qr cmd moyenne	Entier long	507
qr cmd nombre	Entier long	510
qr cmd normal	Entier long	511

QR Commandes (suite)

Constante	Type	Valeur
qr cmd nouveau	Entier long	2000
qr cmd objet couleur fond	Entier long	1003
qr cmd objet couleur fond alt	Entier long	1004
qr cmd objet couleur texte	Entier long	1002
qr cmd ouvrir	Entier long	2001
qr cmd palette colonnes	Entier long	2054
qr cmd palette couleurs de fond	Entier long	2052
qr cmd palette opérateurs	Entier long	2051
qr cmd palette standard	Entier long	2053
qr cmd palette style	Entier long	2050
qr cmd somme	Entier long	506
qr cmd souligné	Entier long	502
qr cmd supprimer colonne	Entier long	2601
qr cmd valeurs répétées	Entier long	2604
qr cmd version enregistrée	Entier long	2004

QR Destination de sortie

Commande(s) associée(s) : QR FIXER DESTINATION, QR LIRE DESTINATION.

Constante	Type	Valeur
qr fichier HTML	Entier long	5
qr fichier texte	Entier long	2
qr imprimante	Entier long	1
qr zone 4D Chart	Entier long	4
qr zone 4D View	Entier long	3

QR Encadrements

Commande(s) associée(s) : QR LIRE ENCADREMENTS.

Constante	Type	Valeur
qr encadrement bas	Entier long	8
qr encadrement droit	Entier long	4
qr encadrement gauche	Entier long	1
qr encadrement haut	Entier long	2
qr encadrement horiz intérieur	Entier long	32
qr encadrement vert intérieur	Entier long	16

QR Lignes pour Propriétés

Commande(s) associée(s) : QR FIXER ENCADREMENTS, QR FIXER INFO LIGNE, QR FIXER PROPRIETE TEXTE, QR LIRE ENCADREMENTS, QR Lire info ligne.

Constante	Type	Valeur
qr détail	Entier long	-2
qr entête	Entier long	-4
qr intitulé	Entier long	-1
qr pied de page	Entier long	-5
qr total général	Entier long	-3

QR Opérateurs

Commande(s) associée(s) : QR FIXER DONNEES TOTAUX, QR LIRE DONNEES TOTAUX.

Constante	Type	Valeur
qr écart type	Entier long	32
qr max	Entier long	8
qr min	Entier long	4
qr moyenne	Entier long	2
qr nombre	Entier long	16
qr somme	Entier long	1

QR Propriétés de document

Commande(s) associée(s) : QR FIXER PROPRIETE DOCUMENT, QR Lire propriete document.

Constante	Type	Valeur
qr dialogue d'impression	Entier long	1
qr unité	Entier long	2

QR Propriétés de texte

Commande(s) associée(s) : QR FIXER PROPRIETE TEXTE, QR Lire propriete texte.

Constante	Type	Valeur
qr couleur de fond	Entier long	8
qr couleur de fond alternée	Entier long	9
qr couleur de texte	Entier long	6
qr gras	Entier long	3
qr italique	Entier long	4
qr justification	Entier long	7
qr police	Entier long	1
qr souligné	Entier long	5
qr taille police	Entier long	2

QR Propriétés de zone

Commande(s) associée(s) : QR FIXER PROPRIETE ZONE, QR Lire propriete zone.

Constante	Type	Valeur
qr barre menu	Entier long	1
qr barre outils colonnes	Entier long	6
qr barre outils couleurs	Entier long	5
qr barre outils opérateurs	Entier long	4
qr barre outils standard	Entier long	2
qr barre outils style	Entier long	3
qr menus contextuels	Entier long	7

QR Types d'états

Commande(s) associée(s) : QR FIXER TYPE ETAT, QR Lire type etat.

Constante	Type	Valeur
qr état en liste	Entier long	1
qr état tableau croisé	Entier long	2

Recherches

Commande(s) associée(s) : FIXER DESTINATION RECHERCHE, Lire dernier chemin recherche, Lire dernier plan recherche.

Constante	Type	Valeur
Description format Texte	Entier long	0
Description format XML	Entier long	1
Vers ensemble	Entier long	1
Vers sélection courante	Entier long	0
Vers sélection temporaire	Entier long	2
Vers variable	Entier long	3

Sauvegarde et restitution

Commande(s) associée(s) : LIRE INFORMATION RESTITUTION, LIRE INFORMATION SAUVEGARDE.

Constante	Type	Valeur
Date dernière restitution	Entier long	0
Date dernière sauvegarde	Entier long	0
Date prochaine sauvegarde	Entier long	4
Statut dernière restitution	Entier long	2
Statut dernière sauvegarde	Entier long	2

Signatures système standard

Les signatures Système standard sont des chaînes de 4 caractères désignant des types de fichiers standard, des types de ressources ou encore des types de données stockés notamment dans le Presse-papiers.

Commande(s) associée(s) : AJOUTER DONNEES AU CONTENEUR, ECRIRE PROPRIETES RESSOURCE, LIRE DONNEES CONTENEUR, Lire proprietes ressource, Tester conteneur.

Constante	Type	Valeur
Document image	Chaîne	PICT
Document MIDI Windows	Chaîne	MID
Document son Windows	Chaîne	WAV
Document texte	Chaîne	TEXT
Document vidéo Windows	Chaîne	AVI

SQL

Commande(s) associée(s) : LISTE SOURCES DONNEES, SQL CHARGER ENREGISTREMENT, SQL FIXER OPTION, SQL FIXER PARAMETRE, SQL LIRE OPTION, SQL LOGIN.

Constante	Type	Valeur
Source de données système	Entier long	2
Source de données utilisateur	Entier long	1
SQL Asynchrone	Entier long	1
SQL Jeu de caractères	Entier long	100
SQL Longueur maxi données	Entier long	3
SQL Nombre maxi lignes	Entier long	2
SQL Paramètre entrée	Entier long	1
SQL Paramètre entrée sortie	Entier long	2
SQL Paramètre sortie	Entier long	4
SQL Timeout connexion	Entier long	5
SQL Timeout requête	Entier long	4
SQL Tous les enregistrements	Entier long	-1
SQL_INTERNAL	Chaîne	;DB4D_SQL_LOCAL;

Statut du process

Commande(s) associée(s) : INFORMATIONS PROCESS, Statut du process.

Constante	Type	Valeur
Détruit	Entier long	-1
Dialogue caché	Entier long	6
En attente drapeau interne	Entier long	4
En attente entrée sortie	Entier long	3
En attente événement	Entier long	2
En exécution	Entier long	0
Endormi	Entier long	1
Inexistant	Entier long	-100
Suspendu	Entier long	5

Styles de caractères

Les constantes préfixées _O_ sont obsolètes et ne doivent plus être utilisées.

Commande(s) associée(s) : CHANGER PROPRIETES ELEMENT, CHANGER STYLE, FIXER STYLE LIGNE MENU, Lire style ligne menu.

Constante	Type	Valeur
Gras	Entier long	1
Italique	Entier long	2
Normal	Entier long	0
Souligné	Entier long	4
_O_Condensé	Entier long	32
_O_Etendu	Entier long	64
_O_Ombré	Entier long	16
_O_Relief	Entier long	8

Touches de fonction

Commande(s) associée(s) : APPELER SUR EVENEMENT.

Constante	Type	Valeur
Touche aide	Entier long	5
Touche bas	Entier long	31
Touche début	Entier long	1
Touche droite	Entier long	29
Touche échappement	Entier long	27
Touche entrée	Entier long	3
Touche F1	Entier long	-122
Touche F10	Entier long	-109
Touche F11	Entier long	-103
Touche F12	Entier long	-111
Touche F13	Entier long	-105
Touche F14	Entier long	-107
Touche F15	Entier long	-113
Touche F2	Entier long	-120
Touche F3	Entier long	-99
Touche F4	Entier long	-118
Touche F5	Entier long	-96
Touche F6	Entier long	-97
Touche F7	Entier long	-98
Touche F8	Entier long	-100
Touche F9	Entier long	-101
Touche fin	Entier long	4
Touche gauche	Entier long	28
Touche haut	Entier long	30
Touche page précédente	Entier long	12
Touche page suivante	Entier long	11
Touche retour arrière	Entier long	8
Touche retour chariot	Entier long	13
Touche tab	Entier long	9

Transformation des images

Commande(s) associée(s) : COMBINER IMAGES, TRANSFORMER IMAGE.

Constante	Type	Valeur
Concaténation horizontale	Entier long	1
Concaténation verticale	Entier long	2
Miroir horizontal	Entier long	3
Miroir vertical	Entier long	4
Passage en niveaux de gris	Entier long	101
Recadrage	Entier long	100
Redimensionnement	Entier long	1
Réinitialisation	Entier long	0
Superposition	Entier long	3
Translation	Entier long	2

Type du process

Commande(s) associée(s) : INFORMATIONS PROCESS.

Constante	Type	Valeur
Aucun	Entier long	0
Autre process 4D	Entier long	-10
Autre process utilisateur	Entier long	4
Créé par commande de menu	Entier long	2
Créé par dialogue d'exécution	Entier long	3
Gestionnaire Apple Event	Entier long	-7
Gestionnaire d’événement	Entier long	-8
Gestionnaire d’index	Entier long	-5
Gestionnaire du cache	Entier long	-4
Gestionnaire du port série	Entier long	-6
Process 4D Server interne	Entier long	-18
Process CSM	Entier long	-22
Process d'activité	Entier long	-26
Process de restitution	Entier long	-21
Process de sauvegarde	Entier long	-19
Process de structure	Entier long	-2
Process du fichier d'historique	Entier long	-20
Process du serveur Web	Entier long	-13
Process exécuté sur client	Entier long	-14
Process exécuté sur serveur	Entier long	1
Process exécution méthode SQL	Entier long	-24
Process gestionnaire de clients	Entier long	-31
Process interface serveur	Entier long	-15
Process macro éditeur de méthod	Entier long	-17
Process minuteur interne	Entier long	-25
Process principal	Entier long	-1
Process sur fermeture	Entier long	-16
Process Web 4D Client	Entier long	-12
Process Web avec contexte	Entier long	-11
Process Web sans contexte	Entier long	-3
Tâche externe	Entier long	-9

Type fenetre

Commande(s) associée(s) : Type fenetre.

Constante	Type	Valeur
Fenêtre externe	Entier long	5
Fenêtre flottante	Entier long	14
Fenêtre modale	Entier long	9
Fenêtre normale	Entier long	8

Type index

Commande(s) associée(s) : CREER INDEX, FIXER INDEX.

Constante	Type	Valeur
Index BTree cluster	Entier long	3
Index BTree standard	Entier long	1
Index de mots clés	Entier long	-1
Type index par défaut	Entier long	0

Types champs et variables

Commande(s) associée(s) : DECLARATION SOAP, INSERER COLONNE FORMULE LISTBOX, LIRE PROPRIETES CHAMP, Type.

Constante	Type	Valeur
Est un BLOB	Entier long	30
Est un booléen	Entier long	6
Est un champ alpha	Entier long	0
Est un entier	Entier long	8
Est un entier 64 bits	Entier long	25
Est un entier long	Entier long	9
Est un Float	Entier long	35
Est un numérique	Entier long	1
Est un pointeur	Entier long	23
Est un tableau 2D	Entier long	13
Est un tableau booléen	Entier long	22
Est un tableau chaîne	Entier long	21
Est un tableau date	Entier long	17
Est un tableau entier	Entier long	15
Est un tableau entierlong	Entier long	16
Est un tableau image	Entier long	19
Est un tableau numérique	Entier long	14
Est un tableau pointeur	Entier long	20
Est un tableau texte	Entier long	18
Est un texte	Entier long	2
Est une date	Entier long	4
Est une heure	Entier long	11
Est une image	Entier long	3
Est une sous table	Entier long	7
Est une variable chaîne	Entier long	24
Est une variable indéfinie	Entier long	5

Valeurs pour Actions standard associée

Commande(s) associée(s) : FIXER PROPRIETE LIGNE MENU, LIRE PROPRIETE LIGNE MENU.

Constante	Type	Valeur
Action afficher Presse papiers	Entier long	23
Action ajouter sous enreg	Entier long	14
Action annuler	Entier long	17
Action coller	Entier long	20
Action copier	Entier long	19
Action couper	Entier long	18
Action CSM	Entier long	36
Action dernier enregistrement	Entier long	6
Action dernière page	Entier long	11
Action effacer	Entier long	21
Action enregistrement précédent	Entier long	4
Action enregistrement suivant	Entier long	3
Action modifier sous enreg	Entier long	12
Action ne pas valider	Entier long	1
Action page précédenete	Entier long	9
Action page suivante	Entier long	8
Action préférences	Entier long	32
Action premier enregistrement	Entier long	5
Action première page	Entier long	10
Action quitter	Entier long	27
Action répéter	Entier long	31
Action supprimer enregistrement	Entier long	7
Action supprimer sous enreg	Entier long	13
Action test application	Entier long	26
Action tout sélectionner	Entier long	22
Action valider	Entier long	2
Pas d'action	Entier long	0
Retour au mode Développement	Entier long	35

Web Services (Client)

Commande(s) associée(s) : APPELER WEB SERVICE, FIXER OPTION WEB SERVICE, Lire infos erreur Web Service.

Constante	Type	Valeur
Web Service afficher dial auth	Entier long	4
Web Service code erreur	Entier long	0
Web Service code erreur HTTP	Entier long	2
Web Service compression deflate	Entier long	1
Web Service compression HTTP	Entier long	6
Web Service dynamique	Entier long	0
Web Service effacer infos auth	Entier long	5
Web Service entrée manuel	Entier long	1
Web Service header SOAP	Entier long	2
Web Service manuel	Entier long	3
Web Service message	Entier long	1
Web Service origine erreur	Entier long	3
Web Service SOAP_1_1	Entier long	0
Web Service SOAP_1_2	Entier long	1
Web Service sortie manuel	Entier long	2
Web Service timeout HTTP	Entier long	1
Web Service version SOAP	Entier long	3

Web Services (Serveur)

Commande(s) associée(s) : DECLARATION SOAP, ENVOYER ERREUR SOAP, Lire informations SOAP.

Constante	Type	Valeur
SOAP entrée	Entier long	1
SOAP erreur client	Entier long	1
SOAP erreur serveur	Entier long	2
SOAP nom méthode	Entier long	1
SOAP nom service	Entier long	2
SOAP sortie	Entier long	2

XML

Commande(s) associée(s) : DOM Lire informations XML, SAX Lire noeud XML, SVG EXPORTER VERS IMAGE.

Constante	Type	Valeur
CDATA XML	Entier long	7
Commentaire XML	Entier long	2
Copier source données XML	Entier long	1
Début document XML	Entier long	1
Début élément XML	Entier long	4
Donnée XML	Entier long	6
Encoding	Entier long	4
Entité XML	Entier long	8
Fin document XML	Entier long	9
Fin élément XML	Entier long	5
ID PUBLIC	Entier long	1
ID SYSTEM	Entier long	2
Instruction de traitement XML	Entier long	3
Lire source données XML	Entier long	0
Nom DOCTYPE	Entier long	3
Posséder source données XML	Entier long	2
URI Document	Entier long	6
Version	Entier long	5

Zone de formulaire

Commande(s) associée(s) : FIXER TAQUET IMPRESSION, Imprimer ligne, Lire taquet impression.

Constante	Type	Valeur
Corps formulaire	Entier long	0
Entête formulaire	Entier long	200
Entête formulaire1	Entier long	201
Entête formulaire10	Entier long	210
Entête formulaire2	Entier long	202
Entête formulaire3	Entier long	203
Entête formulaire4	Entier long	204
Entête formulaire5	Entier long	205
Entête formulaire6	Entier long	206
Entête formulaire7	Entier long	207
Entête formulaire8	Entier long	208
Entête formulaire9	Entier long	209
Pied de page formulaire	Entier long	100
Rupture formulaire0	Entier long	300
Rupture formulaire1	Entier long	301
Rupture formulaire2	Entier long	302
Rupture formulaire3	Entier long	303
Rupture formulaire4	Entier long	304
Rupture formulaire5	Entier long	305
Rupture formulaire6	Entier long	306
Rupture formulaire7	Entier long	307
Rupture formulaire8	Entier long	308
Rupture formulaire9	Entier long	309

Zone Web

Commande(s) associée(s) : WA Creer menu historique URL, WA FIXER PREFERENCE, WA LIRE HISTORIQUE URL, WA LIRE PREFERENCE.

Constante	Type	Valeur
wa autoriser applets Java	Entier long	1
wa autoriser JavaScript	Entier long	2
wa autoriser menu contextuel	Entier long	4
wa autoriser plugins	Entier long	3
wa URLs précédents	Entier long	0
wa URLs suivants	Entier long	1

Index des commandes

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W