macromedia®

FLASH™ 5

Guide de référence ActionScript

macromedia

Remerciements

Gestion de projet : Erick Vera

Rédaction : Jody Bleyle, Mary Burger, Louis Dobrozensky, Stephanie Gowin, Marcelle Taylor et Judy Walthers Von Alten

Édition : Peter Fenczik, Rosana Francescato et Ann Szabla

Multimédia : George Brown, John « Zippy » Lehnus et Noah Zilberberg

Conception de l'impression et de l'aide : Chris Basmajian et Noah Zilberberg

Production : Chris Basmajian et Rebecca Godbois

Localisation (gestion de projet) : Yuko Yagi

Localisation (production) : Masayo « Noppe » Noda et Bowne Global Solutions

Remerciements spéciaux : Jeremy Clark, Brian Dister et l'équipe de développement de Flash, Michael Dominguez, Margaret Dumas, Sherri Harte, Yoshika Hedberg, Tim Hussey, Kipling Inscore, Gwenhaël Jacq, Alyn Kelley, Christophe Pouchard, Pete Santangeli, Denise Seymour et l'équipe Qualité Flash, Cyn Taylor et Eric Wittman

Première édition : Septembre 2000

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103
États-Unis

TABLE DES MATIÈRES

CHAPITRE 6

CHAPITRE 7

INTRODUCTION

Mise en route

ActionScript est le langage de programmation de Flash. Vous pouvez utiliser ActionScript pour contrôler des objets dans les animations Flash, pour créer des éléments de navigation et des éléments interactifs et pour étendre les fonctionnalités de Flash afin de créer des animations et des applications web hautement interactives.

Nouveautés du langage ActionScript de Flash 5

Le langage ActionScript de Flash 5 offre de nouvelles fonctionnalités permettant de créer des sites web interactifs avec des jeux, des formulaires et des sondages sophistiqués, ainsi qu'une interactivité en temps réel, comme celle offerte par les systèmes de discussion électronique.

ActionScript inclut de nombreuses nouvelles fonctionnalités et conventions de syntaxe qui le rapprochent du langage de programmation de base JavaScript. Ce manuel explique des concepts de programmation de base comme les fonctions, les variables, les instructions, les opérateurs, les conditionnels et les boucles. Le chapitre 7 de ce manuel, intitulé « Dictionnaire ActionScript », contient une entrée détaillée pour chaque élément ActionScript.

Ce manuel n'a pas pour but d'enseigner la programmation en général. Il existe de nombreuses sources qui fournissent des informations complémentaires sur les concepts de programmation généraux et sur le langage JavaScript.

L'ECMA (European Computers Manufacturers Association) a rédigé un document appelé Spécification ECMA-262, dérivé du langage JavaScript, qui sert de norme internationale pour le langage JavaScript. ActionScript est basé sur la spécification ECMA-262, qui est disponible à l'adresse http://www.ecma.ch.

Netscape DevEdge Online possède un centre de développement JavaScript (http://developer.netscape.com/tech/javascript/index.html) qui contient de la documentation et des articles utiles à la compréhension d'ActionScript. La ressource la plus importante est le Core JavaScript Guide, situé à l'adresse http://developer.netscape.com/docs/manuals/js/core/jsguide/index.htm.

Différences entre ActionScript et JavaScript

Il n'est pas nécessaire de connaître le langage JavaScript pour utiliser et apprendre le langage ActionScript. Cependant, si vous connaissez le langage JavaScript, ActionScript vous sera familier. Les principales différences qui existent entre ActionScript et JavaScript sont les suivantes :

- ActionScript ne prend pas en charge les objets spécifiques aux navigateurs : Document, Fenêtre et Ancre, par exemple.
- ActionScript ne prend pas entièrement en charge tous les objets prédéfinis JavaScript.
- ActionScript prend en charge des constructions syntaxiques non autorisées dans le langage JavaScript (par exemple, les actions `tellTarget` et `ifFrameLoaded` et la syntaxe à barre oblique).
- ActionScript ne prend pas en charge certaines constructions syntaxiques JavaScript, telles que `switch`, `continue`, `try`, `catch`, `throw` et `statement`.
- ActionScript ne prend pas en charge le constructeur JavaScript `Function`.
- Dans ActionScript, l'action `eval` ne peut exécuter que des références variables.
- Dans JavaScript, `toString` de `undefined` est `undefined`. Dans la version 5 de Flash, pour des raisons de compatibilité avec la version 4, `toString` de `undefined` est `" "`.
- Dans JavaScript, l'évaluation de `undefined` dans un contexte numérique a comme résultat `NaN`. Dans la version 5 de Flash, pour des raisons de compatibilité avec la version 4, l'évaluation de `undefined` a comme résultat `0`.
- ActionScript ne prend pas en charge la norme Unicode mais prend en charge les jeux de caractères ISO-8859-1 et Maj-JIS.

Modification du texte

Vous pouvez entrer des scripts directement dans le panneau Actions en mode Expert. Vous pouvez aussi choisir des éléments dans un menu contextuel ou une liste de boîtes d'outils, comme dans la version 4 de Flash.

Syntaxe à point

Vous pouvez utiliser la syntaxe à point et définir les propriétés et les méthodes d'un objet, y compris des instances et des variables d'un clip. Vous pouvez utiliser la syntaxe à point plutôt que la syntaxe à barre oblique utilisée dans la version 4 de Flash. La syntaxe à barre oblique n'est plus utilisée, mais elle est toujours prise en charge par Flash Player.

Types de données

Le langage ActionScript de la version 5 de Flash prend en charge les types de données suivants : chaîne, nombre, booléen, objet et clip. Plusieurs types de données vous permettent d'utiliser différents types d'information dans ActionScript. Par exemple, vous pouvez créer des tableaux courants et des tableaux d'association.

Variables locales

Vous pouvez déclarer des variables locales qui expirent à la fin de la liste d'actions ou de l'appel de fonction. Ceci vous permet de gérer la mémoire et de réutiliser des noms de variable. Les variables de la version 4 de Flash étaient toutes permanentes, même les variables temporaires, telles que les compteurs de boucle restaient dans l'animation tant que celle-ci n'était pas terminée.

Fonctions définies par l'utilisateur

Vous pouvez définir des fonctions avec des paramètres qui renvoient des valeurs. Ceci vous permet de réutiliser des blocs de code dans vos scripts. Dans la version 4 de Flash, vous aviez la possibilité de réutiliser le code à l'aide de l'action `call`, mais vous ne pouviez pas transmettre des paramètres ou renvoyer des valeurs.

Objet prédéfinis

Vous pouvez utiliser des objets prédéfinis pour accéder à certains types d'informations et les manipuler. Voici un petit échantillon des objets prédéfinis :

- L'objet Math inclut un ensemble complet de constantes et de fonctions mathématiques intégrées comme `E` (constante d'Euler), `cos` (cosinus) et `atan` (tangente d'un arc).
- L'objet Date vous permet d'obtenir des informations sur la date et l'heure, quel que soit le système qui exécute Flash Player.
- L'objet Son vous permet d'ajouter des sons à une animation et de les contrôler pendant l'exécution de l'animation. Par exemple, vous pouvez régler le volume (`setVolume`) ou la balance (`setPan`).
- L'objet Souris vous permet de masquer le curseur standard pour vous permettre d'utiliser un curseur personnalisé.
- L'objet MovieClip vous permet de contrôler les clips sans utiliser une action d'encapsulation comme `tellTarget`. Vous pouvez appeler une méthode comme `play`, `loadMovie` ou `duplicateMovieClip` à partir d'un nom d'instance en utilisant la syntaxe à point (par exemple, `monClip.play()`).

Actions de clip

Vous pouvez utiliser l'action `onClipEvent` pour affecter des actions directement aux instances de clip sur la scène. L'action onClipEvent contient des événements comme `load`, `enterFrame`, `mouseMove` et `data` qui vous permettent de créer de nouvelles sortes d'interactivité évoluée.

Nouvelles actions

Vous pouvez utiliser de nouvelles actions comme `do..while` et `for` pour créer des boucles complexes. D'autres nouvelles actions sont implémentées sous forme de méthodes de l'objet MovieClip ; par exemple, `getBounds`, `attachMovie`, `hitTest`, `swapDepths` et `globalToLocal`.

Smart clips

Les smart clips comportent des scripts internes pouvant être modifiés par vous ou par un autre développeur, sans utiliser le panneau Actions. Vous pouvez transmettre des valeurs à un smart clip en utilisant des paramètres de clip que vous pouvez définir dans la Bibliothèque.

Débogueur

Le Débogueur vous permet d'afficher et de modifier des valeurs de variable et de propriété dans une animation exécutée en mode test, dans le lecteur autonome Flash Player ou dans un navigateur web. Ceci vous permet de détecter facilement des problèmes dans votre script ActionScript.

Prise en charge de XML

L'objet prédéfini XML vous permet de convertir le script ActionScript en documents XML et de les transmettre à des applications côté serveur. Vous pouvez aussi utiliser l'objet XML pour charger des documents XML dans une animation Flash et les interpréter. L'objet prédéfini XMLSocket vous permet de créer une connexion serveur permanente pour transmettre des données XML pour des applications en temps réel.

Optimisation

La version 5 de Flash exécute un certain nombre d'optimisations simples sur le code ActionScript afin d'améliorer les performances et préserver la taille du fichier. À la suite de ces optimisations, la version 5 de Flash produit souvent un code d'octet ActionScript plus petit que celui généré par la version 4 de Flash.

Utilisation de l'aide de Flash pour des actions

La version 5 de Flash inclut une aide contextuelle pour chaque action disponible dans le panneau Actions. Pendant que vous créez des scripts, vous pouvez obtenir des informations sur les actions que vous utilisez.

Pour obtenir l'aide sur les actions :

1. Dans le panneau Actions, sélectionnez une action dans la liste de zone d'outils.
2. Cliquez sur le bouton Aide, en haut du panneau.

   La rubrique associée à l'action apparaît dans le navigateur.

CHAPITRE 1
Présentation d'ActionScript

ActionScript, le langage de programmation de Flash, permet d'interagir avec une animation. Vous pouvez concevoir votre animation de telle sorte que les événements utilisateur, les clics de souris ou les frappes de touches par exemple, déclenchent des scripts qui indiqueront à l'animation l'action à effectuer. Ainsi, vous pouvez rédiger un script qui indiquera à Flash de charger différentes animations dans Flash Player selon le bouton de navigation choisi par l'utilisateur.

Envisagez ActionScript comme un outil permettant de créer une animation qui agit exactement selon vos désirs. Il n'est pas nécessaire de connaître toutes les utilisations possibles de cet outil pour commencer à rédiger des scripts ; si vous avez une idée précise, vous pouvez commencer par des actions simples. Vous pourrez ajouter de nouveaux éléments du langage au fur et à mesure que vous les apprendrez pour accomplir des tâches plus complexes.

Ce chapitre présente ActionScript en tant que langage de programmation orienté objet et offre une vue générale des termes ActionScript. Il décompose également un exemple de script de façon à ce que vous puissiez commencer à vous concentrer sur les éléments les plus importants.

De plus, ce chapitre vous présente le panneau Actions dans lequel vous pouvez construire des scripts en sélectionnant les éléments ActionScript.

À propos des scripts d'ActionScript

Vous pouvez commencer à rédiger des scripts simples sans avoir besoin de tout connaître d'ActionScript. Tout ce dont vous avez besoin, c'est d'un objectif ; le reste est affaire de choix judicieux des actions. La meilleure façon de prendre conscience de la facilité d'ActionScript est de créer un script. Les étapes suivantes associent un script à un bouton qui modifie la visibilité d'un clip.

Pour modifier la visibilité d'un clip :

1. Sélectionnez Fenêtre > Bibliothèques communes > Boutons, puis sélectionnez Fenêtre > Bibliothèques communes > Animations. Placez un bouton et un clip sur la scène.
2. Sélectionnez l'occurrence du clip sur la scène et choisissez Fenêtre > Panneaux > Occurrence.
3. Dans le champ Nom, tapez **testMC**.
4. Sélectionnez le bouton sur la scène et choisissez Fenêtre > Actions pour ouvrir le panneau Actions.
5. Dans le panneau Actions sur objets, cliquez sur la catégorie Actions pour l'ouvrir.
6. Double-cliquez sur l'action `setProperty` pour l'ajouter à la liste des actions.
7. Dans le menu contextuel Propriété, choisissez `_visible` (Visibility).
8. Pour le paramètre Cible, tapez **testMC**.
9. Pour le paramètre Valeur, tapez **0**.

   Le code doit avoir cette forme :

```
on (release) {
    setProperty ("testMC", _visible, false);
}
```

10. Choisissez Contrôle > Tester l'animation et cliquez sur le bouton pour faire disparaître le clip.

ActionScript est un langage de programmation orienté objet. Cela signifie que les actions contrôlent les objets lorsqu'un événement particulier survient. Dans ce script, l'événement release (relâchement) de la souris, l'objet est l'occurrence du clip `MC` et l'action est `setProperty`. Lorsque l'utilisateur clique sur le bouton de l'écran, un événement de `release` déclenche un script qui définit la propriété `_visible` de l'objet `MC` sur `false` et rend l'objet invisible.

Vous pouvez utiliser le panneau Actions pour vous aider à élaborer des scripts simples. Pour utiliser toutes les ressources d'ActionScript, il est important de comprendre comment fonctionne le langage : les concepts, les éléments et les règles utilisés par le langage pour organiser les informations et créer des animations interactives.

Cette section traite du fonctionnement d'ActionScript, des concepts fondamentaux des scripts orientés objet, des objets Flash et du déroulement des scripts. Elle décrit également l'endroit où résident les scripts dans une animation Flash.

À propos de la planification et de la correction des scripts

Lorsque vous rédigez des scripts pour une animation complète, la variété et la quantité des scripts peuvent être importantes. Choisir les types d'actions à utiliser, comment structurer efficacement les scripts et où les placer sont des décisions qui demandent une planification et un contrôle délicats, surtout lorsque l'animation devient de plus en plus complexe.

Avant de commencer à rédiger des scripts, formulez le but à atteindre et assimilez ce que vous voulez obtenir. Cette démarche est aussi importante (et tout aussi longue) que de développer des scénarios pour votre travail. Commencez par noter ce qui doit arriver dans l'animation, comme dans cet exemple :

- Je veux créer mon site entièrement avec Flash.
- Les visiteurs du site devront indiquer leur nom qui sera utilisé dans des messages dans l'ensemble du site.
- Le site comprendra une barre de navigation avec des boutons qui feront le lien avec chaque partie du site.
- Lorsqu'un bouton sera sélectionné, la nouvelle section s'ouvrira en fondu au centre de la scène.
- Une scène possède un formulaire de correspondance dans lequel le nom de l'utilisateur sera déjà inscrit.

Lorsque vous savez ce que vous voulez, vous pouvez construire les objets dont vous avez besoin et rédiger les scripts qui contrôleront ces objets.

L'obtention de scripts fonctionnant correctement demande du temps : souvent plus qu'un cycle de rédaction, de tests et de corrections. La meilleure approche consiste à commencer simplement et à tester le travail fréquemment. Lorsqu'une partie du script fonctionne, choisissez Enregistrer sous pour enregistrer une version du fichier (par exemple, myMovie01.fla) et commencez la rédaction de la partie suivante. Cette démarche vous aidera à identifier efficacement les erreurs et consolidera votre script ActionScript au fur et à mesure que vous rédigez des scripts plus complexes.

À propos des scripts orientés objet

Dans les scripts orientés objet, vous organisez les informations sous forme de groupes appelés *classes*. Vous pouvez créer plusieurs occurrences d'une classe appelées *objets,* afin de les utiliser dans vos scripts. Vous pouvez aussi utiliser les classes prédéfinies d'ActionScript ou créer vos propres classes.

Lorsque vous créez une classe, vous définissez l'ensemble des *propriétés* (caractéristiques) et des *méthodes* (comportements) de chaque objet qu'elle crée, de la même façon que vous définissez des objets dans le monde réel. Par exemple, une personne possède des propriétés telles que le sexe, la taille et la couleur des cheveux, et des méthodes telles que parler, marcher et lancer. Dans cet exemple, « personne » est une classe et chaque individu est un objet ou une *occurrence* de cette classe.

Les objets contenus dans ActionScript peuvent contenir des données et peuvent être représentés de manière graphique sur la scène sous forme de clips. Tous les clips sont des occurrences de la classe prédéfinie MovieClip. Chaque occurrence d'un clip contient toutes les propriétés (par exemple, `_height`, `_rotation`, `_totalframes`) et toutes les méthodes (par exemple, `gotoAndPlay`, `loadMovie`, `startDrag`) de la classe MovieClip.

Pour définir une classe, vous créez une fonction spéciale appelée *fonction de construction* ; les classes prédéfinies ont des fonctions de construction déjà définies. Par exemple, si vous souhaitez inclure dans votre animation des informations sur un cycliste (biker), vous pouvez créer une fonction de construction, `Biker`, avec les propriétés `time` et `distance` et la méthode `rate`, qui vous indique la vitesse à laquelle se déplace un cycliste :

```
function Biker(t, d) {
   this.time = t;
   this.distance = d;
}
function Speed() {
   return this.time / this.distance;
}
Biker.prototype.rate = Speed;
```

Ensuite, vous pouvez créer des copies, c'est-à-dire, des occurrences de la classe. Le code suivant crée des occurrences de l'objet `Biker` appelées `emma` et `hamish`.

```
emma = new Biker(30, 5);
hamish = new Biker(40, 5)
```

Les occurrences peuvent également communiquer entre elles. Pour l'objet `Biker`, vous pouvez créer une méthode appelée `shove` qui permet à un cycliste de pousser un autre cycliste (l'occurrence `emma` pourrait appeler sa méthode `shove` si `hamish` se rapproche trop). Pour transmettre des informations à une méthode, vous utilisez des paramètres (arguments) : par exemple, la méthode `shove` pourrait inclure les paramètres *`who`* et *`howFar`*. Dans cet exemple `emma` pousse `hamish` de 10 pixels :

```
emma.shove(hamish, 10);
```

Dans les scripts orientés objet, les classes peuvent échanger entre elles des propriétés et des méthodes selon un ordre spécifique ; cette procédure s'appelle *héritage*. Vous pouvez utiliser l'héritage pour étendre ou redéfinir les propriétés et les méthodes d'une classe. Une classe qui hérite d'une autre classe s'appelle une *sous-classe*. Une classe qui transmet des propriétés et des méthodes à une autre classe s'appelle *superclasse*. Une classe peut être à la fois une sous-classe et une superclasse.

À propos de l'objet Clip

Les classes prédéfinies d'ActionScript s'appellent des *objets*. Chaque objet vous permet d'accéder à un certain type d'information. Par exemple, l'objet Date possède des méthodes, telles que `getFullYear` et `getMonth`, qui vous permettent de lire des informations de l'horloge système. L'objet Son possède des méthodes telles que `setVolume` et `setPan`, qui vous permettent de contrôler un son dans une animation. L'objet Clip possède des méthodes qui vous permettent de contrôler les occurrences d'un clip (par exemple, `play`, `stop` et `getURL`) et d'obtenir et définir des informations sur leurs propriétés (par exemple, `_alpha`, `_framesloaded`, `_visible`).

Les clips sont les objets les plus importants d'une animation Flash, car ils possèdent des scénarios qui s'exécutent indépendamment les unes des autres. Par exemple, si le scénario principal contient une seule image et qu'un clip de cette image comporte 10 images, chaque image du clip poursuivra son exécution. Ceci permet aux occurrences d'agir en tant qu'objets autonomes capables de communiquer entre eux.

Les occurrences de clip ont chacune un nom unique, ce qui vous permet de les cibler avec une action. Par exemple, vous pouvez avoir plusieurs occurrences sur la scène (par exemple, `leftClip` et `rightClip`) et lire une seule occurrence à la fois. Pour affecter une action qui implique la lecture d'une occurrence spécifique, vous devez utiliser son nom. Dans l'exemple suivant, le nom du clip est `leftClip` :

```
leftClip.play();
```

Les noms des occurrences vous permettent aussi de dupliquer, supprimer et faire glisser des clips pendant la lecture de l'animation. L'exemple suivant duplique l'occurrence `cartItem` pour remplir un caddie avec le nombre d'articles achetés :

```
onClipEvent(load) {
    do {
        duplicateMovieClip("cartItem", "cartItem" + i, i);
        i = i + 1;
    } while (i <= numberItemsPur);
}
```

Les clips ont des propriétés dont vous pouvez définir et extraire les valeurs dynamiquement à l'aide d'ActionScript. La modification et la lecture de ces propriétés peuvent altérer l'apparence et l'identité d'un clip, ce qui constitue la clé de la création de l'interactivité. Par exemple, le script suivant utilise l'action `setProperty` pour affecter à la transparence (paramètre alpha) de l'occurrence `navigationBar` la valeur 10 :

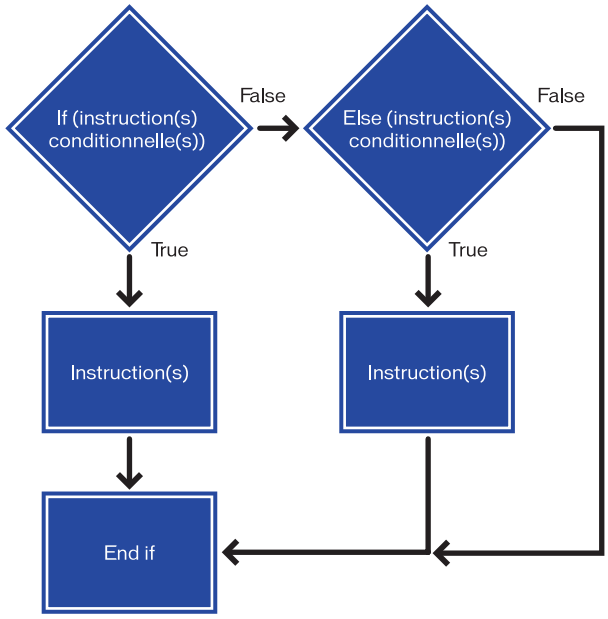
```
setProperty("navigationBar", _alpha, 10);
```

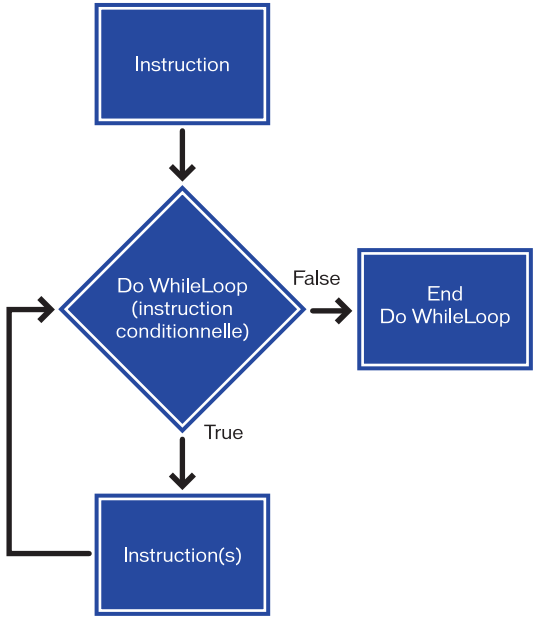
Pour plus d'informations sur d'autres types d'objets, consultez « Utilisation d'objets prédéfinis » à la page 79.

Flux des scripts

ActionScript suit un flux logique. Flash exécute les instructions ActionScript en commençant par la première instruction et poursuit dans l'ordre jusqu'à atteindre la dernière instruction ou celle qui indique à ActionScript d'aller ailleurs.

Parmi les actions qui envoient ActionScript ailleurs qu'à la prochaine instruction figurent les instructions `if`, les boucles `do...while` et l'action `return`.


If (instruction(s) conditionnelle(s))
False
Else (instruction(s) conditionnelle(s))
False
True
True
Instruction(s)
Instruction(s)
End if


Instruction
Do WhileLoop (instruction conditionnelle)
False
End Do WhileLoop
True
Instruction(s)

Une instruction `if` est appelée instruction conditionnelle ou « branche logique » parce qu'elle contrôle le flux d'un script à partir de l'évaluation d'une condition donnée. Par exemple, le code suivant vérifie si la valeur de la variable `number` est inférieure ou égale à 10. Si la valeur `true` est renvoyée (par exemple, la valeur de `number` est 5), la variables `alert` est définit et affiche sa valeur dans un champ de texte saisi comme dans l'exemple suivant :

```
if (number <= 10) {
    alert = "The number is less than or equal to 10";
}
```

Vous pouvez aussi ajouter des instructions `else` pour créer une instruction conditionnelle plus complexe. Dans l'exemple suivant, si la condition renvoie la valeur `true` (par exemple, la valeur de `number` est 3), l'instruction entre la première paire d'accolades s'exécute et la variables `alert` est définie sur la seconde ligne. Si la condition renvoie la valeur `false` (par exemple, la valeur de `number` est 30), le premier bloc de code est ignoré et l'instruction entre accolades après l'instruction `else` s'exécute, comme dans l'exemple suivant :

```
if (number <= 10) {
    alert = "The number is less than or equal to 10";
} else {
    alert = "The number is greater than 10";
}
```

Pour plus d'informations, consultez « Utilisation des instructions if » à la page 71.

Les boucles répètent une action un certain nombre de fois ou jusqu'à ce qu'une condition spécifique s'applique. Dans l'exemple suivant, un clip est dupliqué cinq fois :

```
i = 0;
do {
    duplicateMovieClip ("myMovieClip", "newMovieClip" + i, i);
    newName = eval("newMovieClip" + i);
   setProperty(newName, _x, getProperty("myMovieClip", _x) + (i *
5));
    i = i + 1;
} while (i <= 5);
```

Pour plus d'informations, consultez « Répétition d'une action » à la page 72.

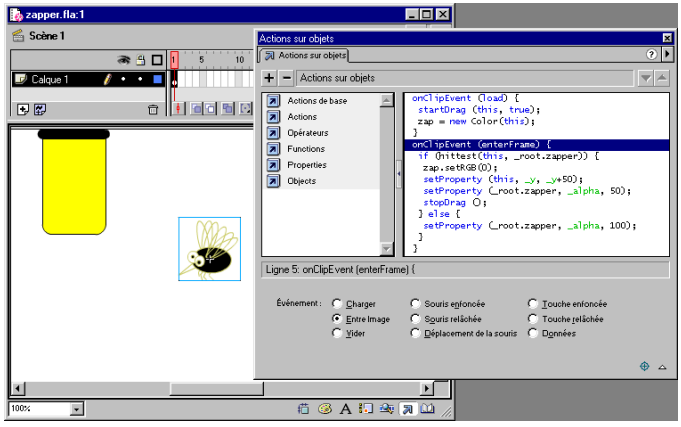
Contrôle de l'exécution d'ActionScript

Lorsque vous rédigez un script, vous utilisez le panneau Actions. Le panneau Actions vous permet d'associer le script à une image du scénario principal ou du scénario d'un clip ou à un bouton ou un clip de la scène.

Flash exécute des actions à des moments différents, en fonction de l'élément auquel elles ont associées :

- Les actions associées à une image sont exécutées lorsque la tête de lecture entre dans cette image.
- Les actions associées à un bouton sont délimitées par une action de gestionnaire `on`.
- Les actions associées à un clip sont insérées dans une action de gestionnaire `onClipEvent`.

Les actions `onClipEvent` et `on` sont appelées gestionnaires, car elles « gèrent » un événement (une occurrence comme un mouvement de souris, la frappe d'une touche ou le chargement d'un clip). Les actions des clips et des boutons s'exécutent lorsque l'événement spécifié par le gestionnaire se produit. Vous pouvez associer plusieurs gestionnaires à un objet si vous souhaitez que différents événements déclenchent les actions. Pour plus d'informations, reportez-vous au chapitre 3, « Création de l'interactivité avec ActionScript ».



Plusieurs gestionnaires `onClipEvent` associés à un clip sur la scène.

Terminologie ActionScript

Comme tout langage de programmation, ActionScript utilise une terminologie spécifique qui respecte certaines règles de syntaxe. La liste ci-dessous présente les principaux termes ActionScript par ordre alphabétique. Ces termes et leur syntaxe sont expliqués en détail dans le chapitre 2, « Rédaction de scripts avec ActionScript ».

Les **actions** sont des instructions qui demandent à une animation de faire quelque chose pendant son exécution. Par exemple, `gotoAndStop` envoie la tête de lecture à une image ou une étiquette spécifique. Dans cet ouvrage, les termes *action* et *instruction* sont interchangeables.

Les **arguments**, également appelés paramètres, sont des espaces réservés qui vous permettent de transmettre des valeurs aux fonctions. Par exemple, la fonction suivante, appelée `welcome`, utilise deux valeurs qu'elle reçoit dans les arguments `firstName` et `hobby` :

```
function welcome(firstName, hobby) {
   welcomeText = "Hello, " + firstName + "I see you enjoy " +
hobby;
}
```

Les **chemins cibles** sont des adresses hiérarchiques des noms des occurences d'un clip, des variables et des objets d'une animation. Vous pouvez nommer une occurrence d'un clip dans le panneau Occurrence. Le scénario principal porte toujours le nom `_root`. Vous pouvez utiliser le chemin cible pour diriger une action sur un clip ou pour obtenir ou définir la valeur d'une variable. Par exemple, l'instruction suivante est le chemin cible de la variable `volume` dans le clip `stereoControl` :

```
_root.stereoControl.volume
```

Les **classes** sont des types de données que vous pouvez créer pour définir un nouveau type d'objet. Pour définir une classe d'objet, vous créez une fonction de construction.

Les **constantes** sont des éléments invariables. Par exemple, la constante `TAB` a toujours la même signification. Les constantes sont utiles pour comparer des valeurs.

Les **événements** sont des actions qui se déroulent pendant l'exécution d'une animation. Par exemple, différents événements sont générés lorsqu'un clip est chargé, lorsque la tête de lecture entre dans une image, lorsque l'utilisateur clique sur un bouton ou un clip ou lorsqu'il frappe sur le clavier.

Les **expressions** sont les parties d'une instruction qui génèrent une valeur. Par exemple, `2 + 2` est une expression.

Les **fonctions** sont des blocs de code réutilisables qui peuvent recevoir des arguments (paramètres) et renvoyer une valeur. Par exemple, la fonction `getProperty` reçoit le nom d'une propriété et le nom de l'occurrence d'un clip et renvoie la valeur de la propriété. La fonction `getVersion` renvoie la version de Flash Player qui est en train de lire l'animation.

Les **fonctions de construction** sont des fonctions que vous utilisez pour définir les propriétés et les méthodes d'une classe. Par exemple, le code suivant crée une nouvelle classe Circle en créant une fonction de construction appelée `Circle` :

```
function Circle(x, y, radius){
   this.x = x;
   this.y = y;
   this.radius = radius;
}
```

Les **gestionnaires** sont des actions spéciales qui « gèrent » un événement comme `mouseDown` ou `load`. Par exemple, `on (onMouseEvent)` et `onClipEvent` sont des gestionnaires ActionScript.

Les **identifiants** sont des noms utilisés pour indiquer une variable, une propriété, un objet, une fonction ou une méthode. Le premier caractère doit être une lettre, un trait de soulignement (_) ou un dollar ($). Chaque caractère qui suit doit être une lettre, un chiffre, un trait de soulignement (_) ou un dollar ($). Par exemple, `firstName` est le nom d'une variable.

Les **méthodes** sont des fonctions affectées à un objet. Une fois que vous avez affecté une fonction, vous pouvez l'appeler en tant que méthode de cet objet. Par exemple, dans le code suivant, `clear` devient une méthode de l'objet `controller` :

```
function Reset(){
   x_pos = 0;
   x_pos = 0;
}
controller.clear = Reset;
controller.clear();
```

Les **mots-clés** sont des mots réservés avec une signification particulière. Par exemple, `var` est un mot-clé utilisé pour déclarer des variables locales.

Les **noms d'occurrence** sont des noms uniques qui vous permettent de cibler des occurrences de clip dans les scripts. Par exemple, un symbole principal de la Bibliothèque pourrait s'appeler `counter` et les deux occurrences de ce symbole dans l'animation pourraient avoir comme noms d'occurrences `scorePlayer1` et `scorePlayer2`. Le code suivant définit une variable appelée `score` à l'intérieur de chaque occurrence du clip en utilisant des noms d'occurrence :

```
_root.scorePlayer1.score += 1
_root.scorePlayer2.score -= 1
```

Les **objets** sont des groupes de propriétés et chaque objet a un nom et une valeur propres. Les objets vous permettent d'accéder à un certain type d'information. Par exemple, l'objet prédéfini Date fournit des informations de l'horloge système.

Les **occurrences** sont des objets qui appartiennent à une certaine classe. Chaque occurrence d'une classe contient toutes les propriétés et les méthodes de cette classe. Tous les clips sont des occurrences avec des propriétés (par exemple, `_alpha` et `_visible`) et des méthodes (par exemple, `gotoAndPlay` et `getURL`) de la classe MovieClip.

Les **opérateurs** sont des termes qui calculent une nouvelle valeur à partir d'une ou de plusieurs valeurs. Par exemple, l'opérateur d'addition (+) additionne deux ou plusieurs valeurs pour en obtenir une nouvelle.

Les **propriétés** sont des attributs qui définissent un objet. Par exemple, `_visible` est une propriété de tous les clips qui définit si ceux-ci sont visibles ou masqués.

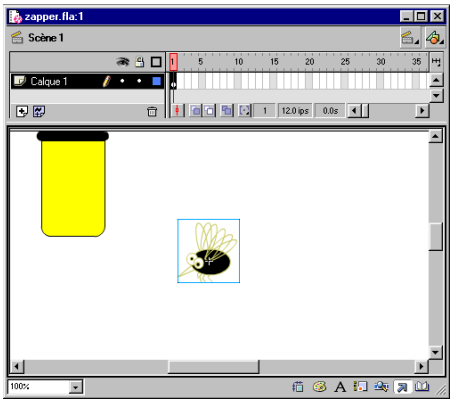
Les **types de données** désignent un ensemble de valeurs et les opérations que vous pouvez effectuer avec elles. Les chaînes, nombres, `true` et `false` (valeurs booléennes), objets et clips sont les types de données d'ActionScript. Pour plus de détails sur ces éléments de langage, consultez « À propos des types de données » à la page 54.

Les **variables** sont des identifiants qui comportent des valeurs de n'importe quel type de données. Vous pouvez créer, modifier et mettre à jour une variable. Vous pouvez extraire les valeurs contenues dans les variables et les utiliser dans des scripts. Dans l'exemple suivant, les identifiants situés à gauche du signe égal sont des variables :

```
x = 5;
name = "Laurent";
customer.address = "66, rue de Paris";
c = new Color(mcinstanceName);
```

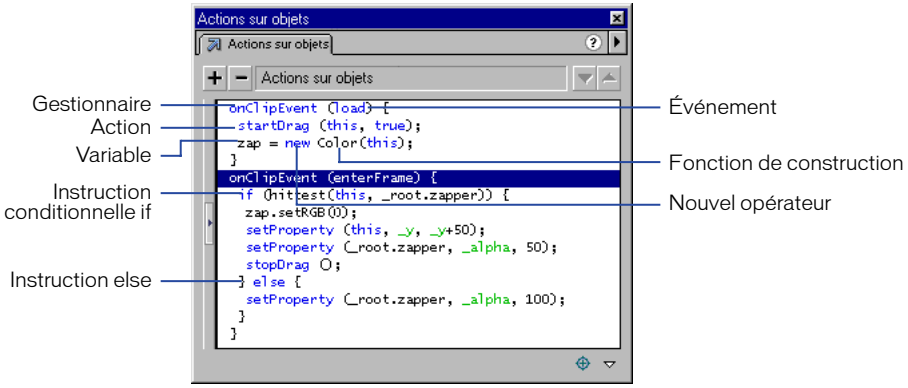
Structure d'un exemple de script

Dans cet exemple d'animation, lorsque l'utilisateur fait glisser l'insecte vers l'insecticide, l'insecte devient noir et tombe, puis l'insecticide clignote. L'animation contient une image et se compose de deux objets, l'occurrence du clip de l'insecte et l'occurrence du clip de l'insecticide. Chaque clip contient à son tour une image.



Occurrences du clip de l'insecte et de l'insecticide sur la scène, à l'image 1.

L'animation contient un seul script associé à l'occurrence `bug (insecte)`, comme dans le panneau Actions sur objet ci-dessous :



Panneau Actions sur objet avec le script associé à l'occurrence `bug`.

Les deux objets doivent être des clips pour que vous puissiez leur affecter des noms d'occurrence dans le panneau Occurrence et les manipuler avec ActionScript. Le nom de l'occurrence insecte est `bug` et le nom de l'occurrence insecticide est `zapper`. Dans le script, l'insecte est nommé `this`, car le script est associé à l'insecte et le mot réservé `this` désigne l'objet qui l'appelle.

Il existe deux gestionnaires `onClipEvent` avec deux événements différents : `load` et `enterFrame`. Les actions contenues dans l'instruction `onClipEvent(load)` s'exécutent une seule fois, lors du chargement de l'animation. Les actions contenues dans l'instruction `onClipEvent(enterFrame)` s'exécutent à chaque fois que la tête de lecture entre dans une image. Même dans une animation composée d'une image, la tête de lecture entre plusieurs fois dans cette image et le script s'exécute à plusieurs reprises. Les actions suivantes se déroulent à l'intérieur de chaque gestionnaire `onClipEvent` :

onClipEvent(load) Une action `startDrag` permet de faire glisser le clip de l'insecte. Une occurrence de l'objet Couleur est créé avec l'opérateur `new` et la fonction de construction Color, `Color`, et est affectée à la variable `zap` :

```
onClipEvent (load) {
   startDrag (this, true);
   zap = new Color(this);
}
```

onClipEvent(enterFrame) Une instruction conditionnelle `if` évalue une action `hitTest` pour vérifier si l'occurrence insecte (`this`) touche ou non l'occurrence insecticide (`_root.zapper`). Cette évaluation peut avoir deux types de résultats, `true` ou `false` :

```
onClipEvent (enterFrame) {
   if (hitTest(_target, _root.zapper)) {
      zap.setRGB(0);
      setProperty (_target, _y, _y+50);
      setProperty (_root.zapper, _alpha, 50);
      stopDrag ();
   } else {
      setProperty (_root.zapper, _alpha, 100);
   }
}
```

Si l'action `hitTest` renvoie la valeur `true`, l'objet `zapper` créé par l'événement `load` est utilisé pour affecter à l'insecte la couleur noire. La propriété *y* de l'insecte (`_y`) se voit affecter sa propre valeur plus `50` pour qu'il tombe. La transparence de l'insecticide (`_alpha`) se voit affecter la valeur `50` afin de l'estomper. L'action `stopDrag` supprime la possibilité de faire glisser l'insecte.

Si l'action `hitTest` renvoie la valeur `false`, l'action qui vient après l'instruction `else` s'exécute et la valeur `_alpha` du piège se voit affecter la valeur `100`. Ceci fait clignoter le piège pendant que sa valeur `_alpha` passe d'un état initial (100) à un état immobile (50) pour revenir à nouveau à son état initial. L'action `hitTest` renvoie la valeur `false` et les instructions `else` s'exécutent une fois que l'insecte a été tué et qu'il est tombé.

Pour voir l'animation, consultez *Aide de Flash*.

Utilisation du panneau Actions

Le panneau Actions vous permet de créer et de modifier des actions pour un objet ou une image en utilisant deux modes de modification différents. Vous pouvez sélectionner des actions déjà rédigées dans la liste des boîtes à outils, glisser et déplacer des actions, et utiliser des boutons pour supprimer ou réorganiser des actions. En mode Normal, vous pouvez rédiger des actions en utilisant des champs de paramètres (arguments) qui vous demandent de spécifier les arguments appropriés. En mode Expert, vous pouvez rédiger et modifier des actions directement dans une zone de texte, de la même façon que vous rédigez un script avec un éditeur de texte.

Pour afficher le panneau Actions :

Choisissez Fenêtre > Actions.

Lorsque vous sélectionnez une occurrence d'un bouton ou d'un clip, le panneau Actions devient actif. Le titre du panneau Actions change pour devenir Actions sur objet si vous sélectionnez un bouton ou un clip, ou Actions sur l'image si vous sélectionnez une image.

Pour sélectionner un mode d'édition :

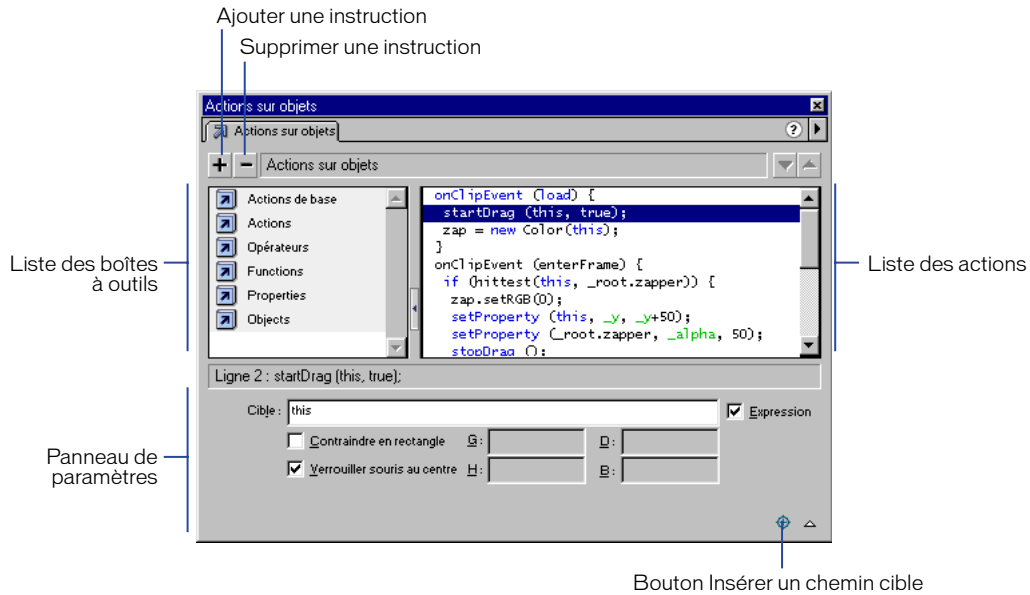
1. Avec le panneau Actions affiché, cliquez sur la flèche située dans l'angle supérieur droit du panneau pour afficher le menu déroulant.
2. Dans le menu déroulant, choisissez Mode Normal ou Mode Expert.

   Chaque script conserve son mode. Par exemple, vous pouvez rédiger un script pour une occurrence d'un bouton en mode Normal et un autre en mode Expert. Le fait de basculer entre les modes du bouton sélectionné provoque le basculement entre les modes du panneau.

Mode Normal

En mode Normal, vous créez des actions en les sélectionnant dans une liste située à gauche du panneau, appelée liste des boîtes à outils. La liste des boîtes à outils contient les catégories Actions de base, Actions, Opérateurs, Fonctions, Propriétés et Objets. La catégorie Actions de base contient les actions Flash les plus simples et n'est disponible qu'en mode Normal. Les actions sélectionnées s'affichent du côté droit du panneau, dans la liste des actions. Vous pouvez ajouter, supprimer ou modifier l'ordre des instructions d'action ; vous pouvez aussi entrer des paramètres (arguments) pour les actions dans les champs prévus à cet effet en bas du panneau.

En mode Normal, vous pouvez utiliser les contrôles du panneau Actions pour supprimer ou modifier l'ordre des instructions dans la liste des actions. Ces contrôles sont particulièrement utiles pour la gestion des actions d'image ou de bouton comportant plusieurs instructions.



Panneau Actions en mode Normal.

Pour sélectionner une action :

1. Cliquez sur une catégorie d'Actions dans la boîte à outils pour afficher les actions correspondant à cette catégorie.
2. Double-cliquez sur une action ou faites-la glisser jusqu'à la fenêtre Script.

Pour utiliser les champs de paramètres :

1. Cliquez sur le bouton Paramètres dans l'angle inférieur droit du panneau Actions pour afficher les champs.
2. Sélectionnez une action et entrez de nouvelles valeurs dans les champs de paramètres pour modifier les paramètres des actions existantes.

Pour insérer un chemin cible pour un clip :

1. Cliquez sur le bouton Chemin cible dans l'angle inférieur droit du panneau Actions pour afficher la boîte de dialogue Insérer un chemin cible.
2. Sélectionnez un clip dans la liste.

Pour monter ou descendre une instruction dans la liste :

1. Sélectionnez une instruction dans la liste des actions.
2. Cliquez sur les boutons flèche Haut ou flèche Bas.

Pour supprimer une action :

1. Sélectionnez une instruction dans la liste des actions.
2. Cliquez sur le bouton Supprimer (-).

Pour modifier les paramètres des actions existantes :

1. Sélectionnez une instruction dans la liste des actions.
2. Entrez de nouvelles valeurs dans les champs de paramètres.

Pour redimensionner la liste des boîtes à outils ou des actions, exécutez l'une des opérations suivantes :

- Faites glisser la barre de fractionnement verticale qui apparaît entre la liste des boîtes à outils et des actions.
- Double-cliquez sur la barre de fractionnement pour réduire la liste des boîtes à outils et double-cliquez encore une fois sur la barre pour afficher à nouveau la liste.
- Cliquez sur le bouton flèche gauche ou flèche droite de la barre de fractionnement pour développer ou réduire la liste.

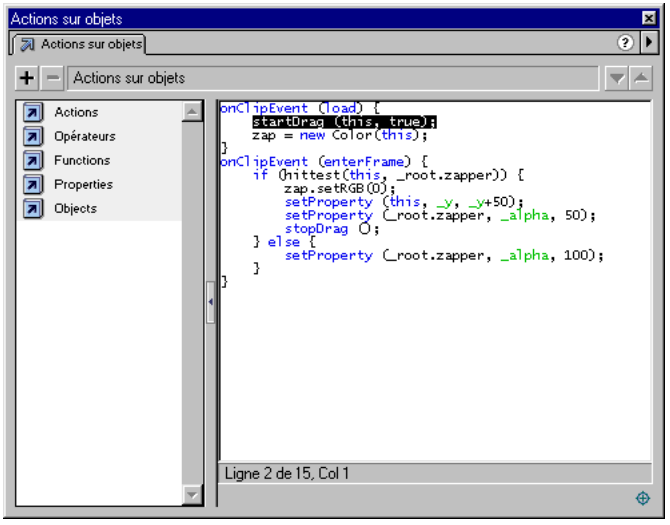
Lorsque la liste des boîtes à outils est masquée, vous pouvez toujours accéder à ses éléments en utilisant le bouton Ajouter (+), situé à l'angle supérieur gauche du panneau Actions.

Mode Expert

En mode Expert, vous pouvez créer des actions en tapant du script ActionScript dans la zone de texte située du côté droit du panneau ou en sélectionnant des actions dans la liste des boîtes à outils située à gauche. Vous modifiez des actions, entrez des paramètres pour les actions ou supprimez des actions directement dans la zone de texte, de la même façon que vous créez un script dans un éditeur de texte.

Le mode Expert permet aux utilisateurs avertis d'ActionScript de modifier leurs scripts à l'aide d'un éditeur de texte, comme c'est le cas avec JavaScript ou VBScript. Les différences entre le mode Expert et le mode Normal sont les suivantes :

- Lorsque vous sélectionnez un élément dans le menu contextuel Ajouter ou la liste des boîtes à outils, l'élément est inséré dans la zone d'édition du texte.
- Aucun champ de paramétrage ne s'affiche.
- Dans le panneau de boutons, seul le bouton Ajouter (+) fonctionne.
- Les boutons Flèche haut et Flèche bas restent inactifs.



Panneau Actions en mode Expert.

Basculement entre les modes de modification

Le fait de changer de mode d'édition pendant la rédaction d'un script peut changer sa mise en forme. Par conséquent, il est préférable d'utiliser un seul mode d'édition par script.

Lorsque vous passez du mode Normal au mode Expert, le retrait et la mise en forme sont conservés. Même si vous pouvez convertir des scripts créés en mode Normal avec des erreurs en scripts en mode Expert, vous ne pouvez pas exporter les scripts tant que les erreurs n'ont pas été corrigées.

Le passage du mode Expert au mode Normal est un peu plus complexe :

- Lorsque vous passez au mode Normal, Flash reformate le script et supprime tous les espaces et les retraits que vous avez ajoutés.
- Si vous passez au mode Normal et que vous revenez au mode Expert, Flash reformate le script par rapport à son apparence en mode Normal.
- Les scripts en mode Expert contenant des erreurs ne peuvent pas être exportés ou convertis en mode Normal et si vous essayez de convertir le script, un message d'erreur s'affiche.

Pour basculer entre les modes d'édition :

Choisissez Mode Normal ou Mode Expert dans le menu déroulant, à l'angle supérieur droit du panneau Actions. Une coche indique le mode sélectionné.

Pour définir vos préférences dans un mode d'édition :

1. Sélectionnez Édition > Préférences.
2. Sélectionnez l'onglet Généralités.
3. Choisissez Dans le panneau Actions, sélectionnez Mode Normal ou Mode Expert dans le menu déroulant.

Utilisation d'un éditeur externe

Même si le mode Expert du panneau Actions vous offre un meilleur contrôle lors de l'édition d'ActionScript, vous pouvez aussi modifier un script sans utiliser Flash. Vous pouvez utiliser ensuite l'action `include` pour ajouter les scripts que vous avez rédigés dans un éditeur externe à un script Flash.

Par exemple, l'instruction suivante importe un fichier de script :

```
#include "externalfile.as"
```

Le texte du fichier de script remplace l'action `include`. Le fichier texte doit être présent lorsque vous exportez l'animation.

Pour ajouter les scripts rédigés dans un éditeur externe à un script Flash :

1. Faites glisser l'action `include` depuis la liste des boîtes à outils jusqu'à la fenêtre Script.
2. Entrez le chemin du fichier externe dans la zone Chemin.

   Vous devez spécifier le chemin par rapport au fichier FLA. Par exemple, si les fichiers myMovie.fla et externalfile.as se trouvent dans le même dossier, le chemin sera externalfile.as. Si externalfile.as est situé dans un sous-dossier nommé scripts, le chemin sera scripts/externalfile.as.

Choix des options du panneau Actions

Le panneau Actions vous permet de travailler avec des scripts de différentes façons. Vous pouvez modifier la taille de la police dans la fenêtre Script. Vous pouvez importer un fichier texte contenant un script ActionScript dans le panneau Actions et exporter des actions sous forme de fichier texte, rechercher et remplacer du texte dans un script, et mettre en surbrillance la syntaxe pour améliorer la lisibilité des scripts et faciliter l'identification des erreurs. Le panneau Actions affiche des avertissements en surbrillance pour les erreurs de syntaxe et les incompatibilités de version avec Flash Player. Il met également en surbrillance les éléments ActionScript *désapprouvés* ou déconseillés.

Ces options du panneau Actions sont disponibles à la fois en mode Normal et en mode Expert, sauf mention contraire.

Pour changer la taille de la police dans la fenêtre Script :

1. Dans le menu déroulant, à l'angle supérieur droit du panneau Actions, choisissez Taille de la police.
2. Sélectionnez Petite, Moyenne ou Grande.

Pour importer un fichier texte contenant du script ActionScript :

1. Dans le menu déroulant, à l'angle supérieur droit du panneau Actions, choisissez Importer du fichier.
2. Sélectionnez un fichier texte contenant du script ActionScript, puis cliquez sur Ouvrir.

Remarque : Les scripts comportant des erreurs ne peuvent être importés qu'en mode Expert. Si vous essayez de les importer en mode Normal, un message d'erreur s'affiche.

Pour exporter des actions sous forme de fichier texte :

1. Dans le menu déroulant, à l'angle supérieur droit du panneau Actions, choisissez Exporter comme fichier.
2. Choisissez un endroit pour enregistrer le fichier, puis cliquez sur Enregistrer.

Pour imprimer des actions :

1 Dans le menu déroulant, à l'angle supérieur droit du panneau Actions, choisissez Imprimer.

La boîte de dialogue Imprimer s'affiche.

2 Choisissez Options et cliquez sur Imprimer.

Remarque : Le fichier imprimé n'inclura pas d'information sur le fichier Flash d'origine. Il est conseillé d'inclure ces informations dans une action `comment` à l'intérieur du script.

Pour rechercher du texte dans un script, choisissez une option dans le menu déroulant du panneau Actions :

- Choisissez Atteindre la ligne pour atteindre une ligne spécifique du script.
- Choisissez Rechercher pour rechercher du texte.
- Choisissez Recherche à nouveau pour rechercher à nouveau du texte.
- Choisissez Remplacer pour rechercher et remplacer du texte.

  En mode Expert, l'option Remplacer analyse tout le corps du texte dans un script. En mode Normal, l'option Remplacer recherche et remplace du texte uniquement dans le champ de paramètres de chaque action. Par exemple, vous ne pouvez pas remplacer toutes les actions `gotoAndPlay` par `gotoAndStop` en mode Normal.

Remarque : Utilisez la commande Rechercher ou Remplacer pour rechercher la liste des actions en cours. Pour rechercher du texte dans chaque script d'une animation, utilisez l'Explorateur d'animations. Pour plus d'informations, consultez *Utilisation de Flash*.

Mise en surbrillance et vérification de la syntaxe

La mise en surbrillance de la syntaxe est une procédure qui consiste à identifier des éléments ActionScript par des couleurs spécifiques. Ceci permet d'éviter des erreurs de syntaxe telles que l'emploi incorrect des majuscules dans les mots-clés. Par exemple, si vous écrivez le mot-clé `typeof` sous la forme `typeOf`, il n'apparaîtra pas en bleu et vous pourrez identifier l'erreur. Lorsque la mise en surbrillance de la syntaxe est activée, le texte est mis en surbrillance comme suit :

- Les mots-clés et les identifiants prédéfinis (par exemple, `gotoAndStop`, `play` et `stop`) s'affichent en bleu.
- Les propriétés s'affichent en vert.
- Les commentaires s'affichent en magenta.
- Les chaînes délimitées par des guillemets s'affichent en gris.

Pour activer ou désactiver la mise en surbrillance :

Choisissez Syntaxe en couleur dans le menu déroulant, à l'angle supérieur droit du panneau Actions. Une coche indique que l'option est activée. Tous les scripts de votre animation sont mis en surbrillance.

Il est conseillé de rechercher des erreurs dans la syntaxe d'un script avant d'exporter une animation. Les erreurs sont répertoriées dans la fenêtre de sortie. Vous pouvez exporter une animation contenant des scripts erronés. Cependant, un message vous avertira que les scripts comportant des erreurs n'ont pas été exportés.

Pour rechercher des erreurs dans la syntaxe du script :

Choisissez Vérifier la syntaxe dans le menu déroulant, à l'angle supérieur droit du panneau Actions.

À propos de la mise en évidence des erreurs

En mode Normal, toutes les erreurs de syntaxe sont mises en surbrillance avec un arrière-plan rouge uni dans la fenêtre Script. Ceci permet d'identifier plus facilement les erreurs. Si vous placez le pointeur de la souris sur une action dont la syntaxe est incorrecte, une info-bulle affiche le message d'erreur associé à l'action. Lorsque vous sélectionnez l'action, le message d'erreur s'affiche également dans le titre de volet de la zone Paramètres

En mode Normal, toutes les incompatibilités d'exportation du script ActionScript sont mises en surbrillance avec un arrière-plan jaune uni dans la fenêtre Script. Par exemple, si la version d'exportation de Flash Player est réglée sur Flash 4, le script ActionScript pris en charge uniquement par Flash 5 Player est mis en surbrillance en jaune. La version d'exportation est définie dans la boîte de dialogue Paramètres de publication.

Toutes les actions désapprouvées sont mises en surbrillance avec un arrière-plan vert dans la boîte à outils. Les actions désapprouvées sont mises en surbrillance uniquement lorsque la version d'exportation de Flash est réglée sur Flash 5.

Pour définir la version d'exportation de Flash Player :

1. Choisissez Fichier > Paramètres de publication.
2. Cliquez sur l'onglet Flash.
3. Choisissez une version d'exportation dans le menu contextuel Version.

Remarque : Vous ne pouvez pas désactiver la mise en surbrillance des erreurs de syntaxe.

Pour activer la mise en surbrillance de la syntaxe désapprouvée :

Choisissez Afficher la syntaxe refusée dans le menu déroulant du panneau Actions.

Pour obtenir une liste complète des messages d'erreur, consultez Annexe C, « Messages d'erreur ».

Affectation d'actions aux objets

Vous pouvez affecter une action à un bouton ou un clip pour exécuter une action lorsque l'utilisateur clique sur ce bouton ou place le pointeur dessus, ou lorsqu'un clip est chargé ou arrive à un image spécifique. Vous affectez l'action à une occurrence du bouton ou du clip et les autres occurrences du symbole ne sont pas concernées (pour affecter une action à une image, consultez « Affectation d'actions aux images » à la page 47).

Lorsque vous affectez une action à un bouton, vous devez imbriquer l'action dans un gestionnaire `on(mouse event)` et spécifier les événements de souris ou de clavier qui déclenche l'action. Lorsque vous affectez une action à un bouton en mode Normal, le gestionnaire `on(mouse event)` est automatiquement inséré.

Lorsque vous affectez une action à une animation, vous devez imbriquer l'action dans un gestionnaire `onClipEvent` et spécifier l'événement de clip qui déclenche l'action. Lorsque vous affectez une action à un clip en mode Normal, le gestionnaire `on(mouse event)` est automatiquement inséré.

Les instructions suivantes permettent d'affecter des actions aux objets à l'aide du panneau Actions en mode Normal.

Une fois que vous avez affecté une action, utilisez la commande Contrôle > Tester l'animation pour vous assurer qu'elle fonctionne correctement. La plupart des actions ne fonctionnent pas en mode de modification.

Pour affecter une action à un bouton ou à un clip :

1. Sélectionnez l'occurrence d'un bouton ou d'un clip, puis choisissez Fenêtre > Actions.

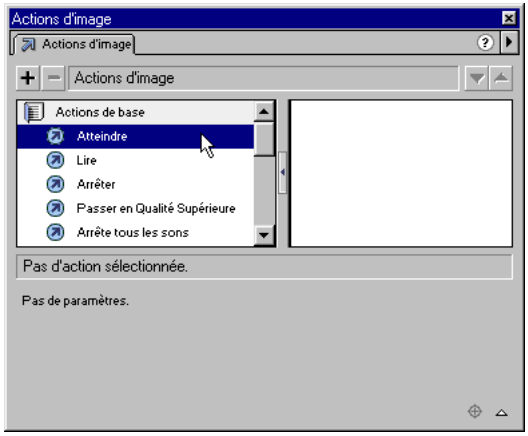
   Si la sélection n'est pas un bouton, une occurrence d'un clip ou une image, ou si la sélection inclut plusieurs objets, le panneau Actions est estompé.

2. Choisissez Mode Normal dans le menu déroulant, à l'angle supérieur droit du panneau Actions sur objets.

3. Pour affecter une action, exécutez l'une des opérations suivantes :

- Cliquez sur le dossier Actions dans la liste des boîtes à outils, du côté gauche du panneau Actions. Double-cliquez sur une action pour l'ajouter à la liste des actions, du côté droit du panneau.
- Faites glisser une action de la liste des boîtes à outils jusqu'à la liste des actions.

- Cliquez sur le bouton Ajouter (+) et choisissez une action dans le menu déroulant.
- Utilisez le raccourci clavier qui apparaît en regard de chaque action dans le menu contextuel.



Sélection d'un objet dans la boîte à outils en mode Normal.

4 Dans les champs de paramètres, en bas du panneau, sélectionnez les paramètres requis pour l'action.

Les paramètres varient en fonction de l'action que vous choisissez. Pour plus d'informations sur les paramètres requis pour chaque action, consultez le chapitre 7, « Dictionnaire ActionScript ». Pour insérer le chemin cible pour un clip dans un champ de paramètres, cliquez sur le bouton Chemin cible, dans l'angle inférieur droit du panneau Actions. Pour plus d'informations, consultez le chapitre 4, « Utilisation des clips ».

5 Répétez les étapes 3 et 4 pour affecter d'autres actions si nécessaire.

Pour tester une action d'objet :

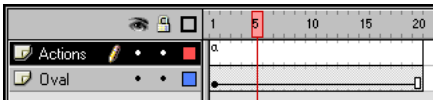
Choisissez Contrôle > Tester l'animation.

Affectation d'actions aux images

Pour faire en sorte qu'une animation exécute une action lorsqu'elle arrive à une image-clé, vous affectez une action d'image à l'image-clé. Par exemple, pour créer une boucle dans le scénario, entre les images 20 et 10, vous ajoutez l'action d'image suivante à l'image 20 :

```
gotoAndPlay (10);
```

Il est conseillé d'insérer les actions d'image dans un calque différent. Les images comportant des actions affichent un petit *a* dans le scénario.



Un « a » dans une image-clé désigne une action d'image.

Une fois que vous avez affecté une action, choisissez Contrôle > Tester l'animation pour vérifier si elle fonctionne correctement. La plupart des actions ne fonctionnent pas en mode de modification.

Les instructions suivantes permettent d'affecter des actions d'image à l'aide du panneau Actions en mode Normal. (Pour plus d'informations sur l'affectation d'une action à un bouton ou un à clip, reportez-vous à la section « Affectation d'une action ou d'une méthode » à la page 125).

Pour affecter une action à une image-clé :

1. Sélectionnez une image-clé dans le scénario, puis choisissez Fenêtre > Actions.

   Si l'image sélectionnée n'est pas une image-clé, l'action est affectée à l'image-clé précédente. Si la sélection n'est pas une image ou si elle contient plusieurs images-clés, le panneau Actions est estompé.

2. Choisissez Mode Normal dans le menu déroulant, à l'angle supérieur droit du panneau Actions sur l'image.

3 Pour affecter une action, exécutez l'une des opérations suivantes :

- Cliquez sur le dossier Actions dans la liste des boîtes à outils, du côté gauche du panneau Actions. Double-cliquez sur une action pour l'ajouter à la liste des actions, du côté droit du panneau.
- Faites glisser une action de la liste des boîtes à outils jusqu'à la liste des actions.
- Cliquez sur le bouton Ajouter (+) et choisissez une action dans le menu déroulant.
- Utilisez le raccourci clavier qui apparaît en regard de chaque action dans le menu contextuel.
- Dans les champs de paramètres, en bas du panneau, sélectionnez les paramètres requis pour l'action.

4 Pour affecter d'autres actions, sélectionnez une autre image-clé et répétez l'étape 3.

Pour tester une action d'image :

Choisissez Contrôle > Tester l'animation.

CHAPITRE 2
Rédaction de scripts avec ActionScript

Lorsque vous créez des scripts dans ActionScript, vous pouvez choisir le niveau de détails à utiliser. Pour des actions simples, vous pouvez utiliser le panneau Actions en mode Normal et créer des scripts en choisissant les options dans les menus et les listes. Cependant, si vous souhaitez utiliser ActionScript pour écrire des scripts plus avancés, vous devez comprendre le fonctionnement d'ActionScript en tant que langage.

Comme les autres langages de programmation, ActionScript contient des composants (objets et fonctions prédéfinis, notamment) et vous permet de créer vos propres objets et fonctions. ActionScript suit ses propres règles de syntaxe, réserve des mots-clés, met à votre disposition des opérateurs et vous permet d'utiliser des variables pour stocker et extraire des informations.

La syntaxe et le style d'ActionScript ressemblent beaucoup à ceux de JavaScript. Flash 5 convertit les scripts ActionScript écrits avec les versions précédentes de Flash.

Utilisation de la syntaxe d'ActionScript

ActionScript possède des règles de grammaire et de ponctuation qui déterminent les caractères ou les mots porteurs de sens et l'ordre dans lequel ils peuvent être écrits. Par exemple, en français, un point termine une phrase. Dans ActionScript, un point-virgule termine une instruction.

Les règles générales suivantes s'appliquent à tous les scripts ActionScript. La plupart des termes ActionScript possèdent également leurs propres exigences ; pour les règles concernant un terme spécifique, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

Syntaxe à point

Dans ActionScript, un point (.) est utilisé pour indiquer les propriétés ou les méthodes associées à un objet ou à un clip. Il est également utilisé pour identifier le chemin cible permettant d'accéder à un clip ou à une variable. Une expression comprenant un point commence par le nom de l'objet ou du clip suivi d'un point et se termine par la propriété, la méthode ou la variable que vous souhaitez spécifier.

Par exemple, la propriété `_x` du clip indique l'axe *x* de positionnement du clip sur la scène. L'expression `ballMC._x` renvoie à la propriété `_x` de l'occurrence `ballMC` du clip.

Dans un autre exemple, `submit` est une variable intégrée au clip `form` qui est imbriqué dans le clip `shoppingCart`. L'expression `shoppingCart.form.submit = true` définit la variable `submit` de l'occurrence `form` sur `true`.

L'expression de la méthode d'un objet ou d'un clip se fait selon le même STOP schéma. Par exemple, la méthode `play` de l'occurrence `ballMC` déplace la tête de lecture dans le scénario de `ballMC`, comme dans l'instruction suivante :

```
ballMC.play();
```

La syntaxe à point utilise également deux alias spéciaux, `_root` et `_parent`. L'alias `_root` renvoie à la chronologie principale. Vous pouvez utiliser l'alias `_root` pour créer un chemin cible absolu. Par exemple, l'instruction suivante appelle la fonction `buildGameBoard` dans le clip `functions` du scénario principal :

```
_root.functions.buildGameBoard();
```

Vous pouvez utiliser l'alias `_parent` pour renvoyer à un clip dans lequel est imbriqué le clip courant. Vous pouvez utiliser `_parent` pour créer un chemin cible relatif. Par exemple, si le clip `dog` est imbriqué dans le clip `animal`, l'instruction suivante de l'occurrence `dog` indique à `animal` de s'arrêter :

```
_parent.stop();
```

Voir chapitre 4, « Utilisation des clips ».

Syntaxe à barre oblique

La syntaxe à barre oblique était utilisée dans Flash 3 et 4 pour indiquer le chemin cible d'un clip ou d'une variable. Cette syntaxe est toujours acceptée parla version 5 de FlashPlayer, mais son utilisation n'est pas recommandée. Dans cette syntaxe, les barres obliques sont utilisées à la place des points pour indiquer le chemin d'un clip ou d'une variable. Pour indiquer une variable, vous la faites précéder de deux points, comme dans l'exemple suivant :

```
myMovieClip/childMovieClip:myVariable
```

Vous pouvez écrire le même chemin cible en utilisant la syntaxe à point :

```
myMovieClip.childMovieClip.myVariable
```

La syntaxe à barre oblique était le plus souvent utilisée avec l'action `tellTarget` dont l'utilisation n'est plus recommandée.

Remarque : l'action `with` est maintenant préférée à `tellTarget` pour des raisons de compatibilité avec la syntaxe à point. Pour plus d'informations, reportez-vous à leurs entrées individuelles dans le chapitre 7, « Dictionnaire ActionScript ».

Accolades

Les instructions ActionScript sont regroupées en blocs par des accolades `({ })`, comme le montre le script suivant :

```
on(release) {
    myDate = new Date();
    currentMonth = myDate.getMonth();
}
```

Voir « Utilisation des actions » à la page 69.

Points-virgules

Une instruction ActionScript est terminée par un point-virgule, mais si vous oubliez celui-ci, Flash continuera de compiler votre script. Par exemple, les instructions suivantes se finissent toutes par des points-virgules :

```
column = passedDate.getDay();
row    = 0;
```

Les mêmes instructions pourraient être écrites sans les points-virgules de fin :

```
column = passedDate.getDay()
row    = 0
```

Parenthèses

Lorsque vous définissez une fonction, placez les arguments entre parenthèses :

```
function myFunction (name, age, reader){
   ...
}
```

Lorsque vous appelez une fonction, incluez tous les arguments transmis à la fonction entre parenthèses, comme indiqué :

```
myFunction ("Steve", 10, true);
```

Vous pouvez également utiliser les parenthèses pour supplanter l'ordre de priorité ActionScript ou pour faciliter la lecture des instructions ActionScript. Voir « Ordre de priorité des opérateurs » à la page 63.

Les parenthèses servent également à évaluer une expression située à gauche d'un point dans une syntaxe à point. Par exemple, dans l'instruction suivante, les parenthèses obligent à l'évaluation de `new color(this)` et à la création d'un objet de nouvelle couleur :

```
onClipEvent(enterFrame) {
   (new Color(this)).setRGB(0xffffff);
}
```

Si vous n'aviez pas utilisé de parenthèses, vous auriez dû ajouter une instruction au code pour l'évaluer :

```
onClipEvent(enterFrame) {
   myColor = new Color(this);
   myColor.setRGB(0xffffff);
}
```

Majuscules et minuscules

Dans ActionScript, seuls les mots-clés sont dépendants de la casse ; pour le reste d'ActionScript, vous pouvez utiliser les majuscules et les minuscules indifféremment. Par exemple, les instructions suivantes sont équivalentes :

```
cat.hilite = true;
CAT.hilite = true;
```

Cependant, il est intéressant d'employer les majuscules selon des conventions fixes, comme celles utilisées dans cet ouvrage, car cela permet d'identifier plus facilement les noms des fonctions et des variables lors de la lecture du code ActionScript.

Si vous n'utilisez pas la bonne casse dans les mots-clés, votre script comportera des erreurs. Lorsque la syntaxe colorisée est activée dans le panneau Actions, les mots-clés écrits de manière correcte apparaissent en bleu. Pour plus d'informations, voir « Mots-clés » à la page 53 et « Mise en surbrillance et vérification de la syntaxe » à la page 43.

Commentaires

Dans le panneau Actions, utilisez l'instruction `comment` pour ajouter des remarques à une image ou à un bouton d'action lorsque vous souhaitez conserver une trace de l'objectif de l'action. Les commentaires sont également utiles lorsque les informations sont transmises à d'autres développeurs, si vous travaillez avec des collaborateurs, ou si vous fournissez des extraits de code.

Lorsque vous choisissez l'action `comment`, les signes `//` sont insérés dans le script. Même un script simple est plus facile à comprendre si vous l'annotez lors de sa création :

```
on(release) {
    // create new Date object
    myDate = new Date();
    currentMonth = myDate.getMonth();
    // convert month number to month name
    monthName = calcMonth(currentMonth);
    year = myDate.getFullYear();
    currentDate = myDate.getDat ();
}
```

Les commentaires apparaissent en rose dans la fenêtre Script. Ils peuvent avoir n'importe quelle longueur sans que cela affecte la taille du fichier exporté et ils ne suivent aucune règle de syntaxe ou de mots-clés relative à ActionScript.

Mots-clés

ActionScript réserve des mots pour une utilisation spécifique à ce langage, si bien que vous ne pouvez pas les utiliser comme variable, fonction ou noms d'étiquette. Le tableau suivant répertorie tous les mots-clés ActionScript :

`break`	`for`	`new`	`var`
`continue`	`function`	`return`	`void`
`delete`	`if`	`this`	`while`
`else`	`in`	`typeof`	`with`

Pour plus d'informations sur un mot-clé spécifique, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

Constantes

Une constante est une propriété dont la valeur ne varie jamais. La liste des constantes apparaît en lettres majuscules dans la barre d'outils Actions et dans le chapitre 7, « Dictionnaire ActionScript ».

Par exemple, les constantes `BACKSPACE`, `ENTER`, `QUOTE`, `RETURN`, `SPACE` et `TAB` sont des propriétés de l'objet Key et renvoient aux touches du clavier. Pour savoir si un utilisateur appuie sur la touche Entrée, utilisez l'instruction suivante :

```
if(keycode() == Key.ENTER) {
   alert = "Are you ready to play?"
   controlMC.gotoAndStop(5);
}
```

À propos des types de données

Un type de données décrit le genre d'informations qu'une variable ou qu'un élément ActionScript peut contenir. Il existe deux sortes de types de données : le type primitif et le type référence. Les types primitifs de données (chaîne, nombre et booléen) ont une valeur constante et peuvent donc contenir la valeur courante de l'élément qu'ils représentent. Les types références de données (clip et objet) possèdent des valeurs modifiables et contiennent donc les références à la valeur courante de l'élément. Les variables contenant des types primitifs de données ont un comportement différent de celles contenant des types références dans certaines situations. Voir « Utilisation des variables dans un script » à la page 60.

Chaque type de données possède ses propres règles et est décrit ci-dessous. Les références sont incluses pour les types de données traités en détail ci-après.

Chaîne

Une chaîne est une séquence de caractères (lettres, chiffres et signes de ponctuation, par exemple). Vous insérez des chaînes dans une instruction ActionScript en les plaçant entre des guillemets simples ou doubles. Les chaînes sont traitées comme des caractères et non comme des variables. Par exemple, dans l'instruction suivante, « **L7** » est une chaîne :

```
favoriteBand = "L7";
```

Vous pouvez utiliser l'opérateur d'addition (+) pour *concaténer* ou joindre deux chaînes. ActionScript traite les espaces au début ou à la fin d'une chaîne comme faisant partie de la chaîne. L'expression suivante contient un espace après la virgule :

```
greeting = "Welcome, " + firstName;
```

Même si ActionScript n'est pas dépendant de la casse pour les références aux variables, aux noms d'occurrence et aux libellés d'images, les chaînes de caractères distinguent, elles, la casse. Par exemple, les deux instructions suivantes placent des textes différents dans les champs de texte spécifiés car **"Bonjour"** et **"BONJOUR"** sont des chaînes de caractères.

```
invoice.display = "Hello";
invoice.display = "HELLO";
```

Pour inclure un guillemet dans une chaîne, il faut le faire précéder d'une barre oblique inverse (\). Cette opération s'appelle « échapper » un caractère. D'autres caractères ne peuvent pas être représentés dans ActionScript sans l'emploi de séquences d'échappement particulières. Le tableau suivant répertorie l'ensemble des caractères d'échappement d'ActionScript :

Séquence d'échappement	Caractère
`\b`	Caractère de retour arrière (ASCII 8)
`\f`	Caractère de changement de page (ASCII 12)
`\n`	Caractère de changement de ligne (ASCII 10)
`\r`	Caractère de retour chariot (ASCII 13)
`\t`	Caractère de tabulation (ASCII 9)
`\"`	Guillemet double
`\'`	Guillemet simple
`\\`	Barre oblique inverse
`\000 - \377`	Un octet spécifié en octal
`\x00 - \xFF`	Un octet spécifié en hexadécimal
`\u0000 - \uFFFF`	Un caractère unicode 16 bits spécifié en hexadécimal

Nombre

Ce type de données correspond à un nombre à virgule flottante à double précision. Vous pouvez manipuler les nombres avec les opérateurs arithmétiques d'addition (+), de soustraction (-), de multiplication (*), de division (/), de pourcentage (%), d'incrémentation (++) et de décrémentation (--). Vous pouvez également utiliser les méthodes de l'objet prédéfini Math pour manipuler les nombres. L'exemple suivant utilise la méthode `sqrt` (racine carrée) pour renvoyer la racine carrée du nombre 100 :

```
Math.sqrt(100);
```

Voir « Opérateurs numériques » à la page 64.

Booléen

Une valeur booléenne est soit `true`, soit `false`. ActionScript convertit également les valeurs `true` et `false` en 1 et 0 lorsque cela est nécessaire. Les valeurs booléennes sont le plus souvent utilisées dans les instructions ActionScript effectuant des comparaisons pour contrôler le déroulement d'un script. Par exemple, dans le script suivant, l'animation est lue si la variable `password` est `true` :

```
onClipEvent(enterFrame) {
    if ((userName == true) && (password == true)){
        play();
    }
}
```

Voir « Utilisation des instructions if » à la page 71 et « Opérateurs logiques » à la page 65.

Objet

Un objet est un groupe de propriétés. Chaque propriété possède un nom et une valeur. La valeur d'une propriété peut être de n'importe quel type de données Flash, même un type de données objet. Cela vous permet d'arranger les objets les uns dans les autres, ou de les « imbriquer ». Pour spécifier les objets et leurs propriétés, vous devez utiliser le signe du point (.). Par exemple, dans le code suivant, `hoursWorked` est une propriété de `weeklyStats`, qui est une propriété de `employee` :

```
employee.weeklyStats.hoursWorked
```

Vous pouvez utiliser les objets ActionScript prédéfinis pour localiser et manipuler des genres spécifiques d'informations. Par exemple, l'objet Math possède des méthodes qui effectuent des opérations mathématiques sur les nombres que vous leur transmettez. Cet exemple utilise la méthode `sqrt` :

```
squareRoot = Math.sqrt(100);
```

L'objet MovieClip d'ActionScript possède des méthodes qui vous permettent de contrôler les occurrences des symboles de clips sur la scène. Cet exemple utilise les méthodes `play` et `nextFrame` :

```
mcInstanceName.play();
mc2InstanceName.nextFrame();
```

Vous pouvez également créer vos propres objets afin d'organiser les informations de vos animations. Pour ajouter de l'interactivité à votre animation avec ActionScript, vous aurez besoin d'un certain nombre d'informations : par exemple, vous pourrez avoir besoin d'un nom d'utilisateur, de la vitesse d'une balle, des noms des objets contenus dans un caddie, du nombre d'images chargées, du code postal de l'utilisateur et de la dernière touche utilisée. La création d'objets personnalisés vous permet d'organiser ces informations, de simplifier la rédaction du script et de réutiliser vos scripts. Pour plus d'informations, voir « Utilisation des objets personnalisés » à la page 83.

Clips

Les clips sont des symboles qui peuvent s'animer dans une animation Flash. Ils correspondent au seul type de données renvoyant à un élément graphique. Le type de données clip vous permet de contrôler les symboles des clips en utilisant les méthodes de l'objet MovieClip. Vous appelez les méthodes en utilisant le signe du point (.), par exemple :

```
myClip.startDrag(true);
parentClip.childClip.getURL( "http://www.macromedia.com/support/"
+ produit);
```

À propos des variables

Une variable est un conteneur qui stocke des informations. Le conteneur reste toujours le même, c'est le contenu qui peut varier. En modifiant la valeur d'une variable pendant la lecture d'une animation, vous pouvez enregistrer et sauvegarder les informations relatives aux actions de l'utilisateur, enregistrer les valeurs modifiées pendant la lecture de l'animation ou évaluer si une condition donnée prend la valeur true ou false.

La première fois que vous définissez une variable, affectez-lui toujours une valeur connue. Cela permet d'initialiser la variable et est souvent effectué dans la première image de l'animation. L'initialisation des variables facilite le suivi et la comparaison de la valeur de la variable pendant la lecture de l'animation.

Les variables peuvent contenir tous les types de données : nombre, chaîne, booléen, objet ou clip. Le type de données contenu dans une variable affecte la façon dont est modifiée la valeur de la variable lorsqu'elle est affectée à un script.

Les types d'informations standard que vous pouvez stocker dans une variable comprennent une URL, un nom d'utilisateur, le résultat d'une opération mathématique, le nombre de fois qu'un événement s'est produit ou si un bouton a été actionné. Chaque occurrence d'animation ou de clip possède son propre jeu de variables, chaque variable possédant sa propre valeur indépendante des variables des autres animations ou clips.

Dénomination d'une variable

Le nom d'une variable doit suivre les règles suivantes :

- Il doit s'agir d'un identifiant.
- Il ne peut pas s'agir d'un mot-clé ni un littéral Booléen (`true` ou `false`).
- Il doit être unique dans son étendue. Voir « Portée d'une variable » à la page 59.

Définition du type d'une variable

Avec Flash, il n'est pas nécessaire de définir explicitement le type de données contenues dans la variable. Flash détermine le type de données de la variable lorsque celle-ci est affectée :

```
x = 3;
```

Dans l'expression `x = 3`, Flash évalue l'élément à droite de l'opérateur et détermine qu'il s'agit du type nombre. Une affectation ultérieure pourra changer le type de `x` ; par exemple, `x = "hello"` modifie le type de `x` en chaîne. Une variable à laquelle aucune valeur n'a été affectée est du type non défini `undefined`.

ActionScript convertit automatiquement les types de données lorsqu'une expression le nécessite. Par exemple, lorsque vous transmettez une valeur à l'action `trace`, `trace` convertit automatiquement la valeur en chaîne et l'envoie à la fenêtre Résultat. Dans les expressions avec opérateurs, ActionScript convertit les types de données en fonction des besoins, par exemple, lors de l'utilisation dans une chaîne, l'opérateur + s'attend à ce que l'autre opérande soit une chaîne :

```
"Next in line, number " + 7
```

ActionScript convertit le nombre 7 en chaîne `"7"` et l'ajoute à la fin de la première chaîne, ce qui aboutit à la chaîne suivante :

```
"Next in line, number 7"
```

Lorsque vous déboguez des scripts, il est souvent utile de déterminer le type de données d'une expression ou d'une variable pour comprendre pourquoi elle se comporte de telle manière. Vous pouvez effectuer cette opération avec l'opérateur `typeof`, comme dans l'exemple suivant :

```
trace(typeof(variableName));
```

Pour convertir une chaîne en valeur numérique, utilisez la fonction `Number`. Pour convertir une valeur numérique en chaîne, utilisez la fonction `String`. Reportez-vous à leurs entrées individuelles dans le chapitre 7, « Dictionnaire ActionScript ».

Portée d'une variable

La « portée » d'une variable fait référence à la zone dans laquelle la variable est connue et peut être référencée. Dans ActionScript, les variables peuvent être globales ou locales. Une variable globale est partagée entre tous les scénarios ; une variable locale n'est disponible que dans son seul bloc de code (entre accolades).

Vous pouvez utiliser l'instruction `var` pour déclarer une variable locale dans un script. Par exemple, les variables `i` et `j` sont souvent utilisées comme compteurs de boucles. Dans l'exemple suivant, `i` est utilisée comme variable locale et existe uniquement dans la fonction `makeDays` :

```
function makeDays(){
   var i
   for( i = 0; i < monthArray[month]; i++ ) {

      _root.Days.attachMovie( "DayDisplay", i, i + 2000 );

      _root.Days[i].num = i + 1;
      _root.Days[i]._x = column * _root.Days[i]._width;
      _root.Days[i]._y = row * _root.Days[i]._height;

      column = column + 1;

      if (column == 7 ) {

         column = 0;
         row = row + 1;
      }
   }
}
```

Les variables locales peuvent également empêcher les conflits de noms qui pourraient provoquer des erreurs dans votre animation. Par exemple, si vous utilisez `name` comme variable locale, vous pouvez l'utiliser pour stocker un nom d'utilisateur dans un contexte et une occurrence de clip dans un autre ; ces variables fonctionnant dans des portées séparées, il n'y aura pas de conflit.

Une pratique intéressante consiste à utiliser des variables locales dans le corps d'une fonction pour que la fonction puisse agir en tant que partie de code indépendante. Une variable locale n'est modifiable qu'au sein de son propre bloc de code. Si une expression dans une fonction utilise une variable globale, un élément extérieur pourra modifier sa valeur, ce qui modifiera la fonction.

Déclaration de variables

Pour déclarer des variables globales, utilisez l'action `setVariables` ou l'opérateur d'affectation (=). Ces deux méthodes obtiennent le même résultat.

Pour déclarer des variables locales, utilisez l'instruction `var` dans le corps de la fonction. Les variables locales ont une portée limitée au bloc et expirent à la fin du bloc. Les variables locales qui ne sont pas déclarées dans un bloc expirent à la fin du script dans lequel elles sont contenues.

Remarque : l'action `call` crée également une nouvelle portée de variable locale pour le script appelé. Lorsque le script appelé s'achève, cette portée de variable locale disparaît. Cependant, cette opération n'est pas recommandée car l'action `call` a été remplacée par l'action `with` qui est plus compatible avec la syntaxe à point.

Pour tester la valeur d'une variable, utilisez l'action `trace` qui enverra la valeur dans la fenêtre Résultat. Par exemple, `trace(hoursWorked)` envoie la valeur de la variable `hoursWorked` à la fenêtre Résultat en mode test d'animation. Vous pouvez également vérifier et placer les valeurs de la variable dans le Débogueur en mode test d'animation. Pour plus d'informations, reportez-vous au chapitre 6, « Dépannage d'ActionScript ».

Utilisation des variables dans un script

Vous devez déclarer une variable dans un script avant de pouvoir l'utiliser dans une expression. Si vous utilisez une variable qui n'est pas déclarée, comme dans l'exemple suivant, la valeur de la variable sera `undefined` et votre script générera une erreur :

```
getURL(myWebSite);
myWebSite = "http://www.shrimpmeat.net";
```

L'instruction déclarant la variable `myWebSite` doit venir en premier de sorte que la variable dans l'action `getURL` puisse être remplacée par une valeur.

Vous pouvez modifier plusieurs fois la valeur d'une variable dans un script. Le type de données contenu dans la variable affecte les conditions et le moment où la variable sera modifiée. Les types primitifs de données, comme les chaînes et les nombres, sont transmis par valeur. Cela signifie que le contenu courant de la variable est transmis à la variable.

Dans l'exemple suivant, `x` est défini sur 15 et cette valeur est copiée dans `y`. Lorsque `x` est modifiée en 30, la valeur de `y` reste 15 car `y` ne va pas chercher sa valeur dans `x` ; elle contient la valeur de `x` qui lui a été transmise.

```
var x = 15;
var y = x;
var x = 30;
```

Dans un autre exemple, la variable `in` contient une valeur primitive, 9, la valeur courante est donc transmise à la fonction `sqrt` et la valeur renvoyée est 3 :

```
function sqrt(x){
  return x * x;
}

var in = 9;
var out = sqr(in);
```

La valeur de la variable `in` ne change pas.

Le type de données object peut contenir tant d'informations complexes qu'une variable de ce type ne contient pas la valeur courante ; elle contient une référence à la valeur. Cette référence est comme un alias qui désigne le contenu de la variable. Lorsque la variable a besoin de connaître sa valeur, la référence demande le contenu et renvoie la réponse sans transférer la valeur à la variable.

Le code suivant est un exemple de transmission par référence :

```
var myArray = ["tom", "dick"];
var newArray = myArray;
myArray[1] = "jack";
trace(newArray);
```

Le code ci-dessus crée un objet Array appelé `myArray` qui contient deux éléments. La variable `newArray` est créée et fait référence à `myArray`. Lorsque le deuxième élément de `myArray` est modifié, cela affecte toutes les variables lui faisant référence. L'action `trace` enverra `["tom", "jack"]` dans la fenêtre Résultat.

Dans l'exemple suivant, `myArray` contient un objet Array qui est transmis à la fonction `zeroArray` par référence. La fonction `zeroArray` modifie le contenu du tableau en `myArray`.

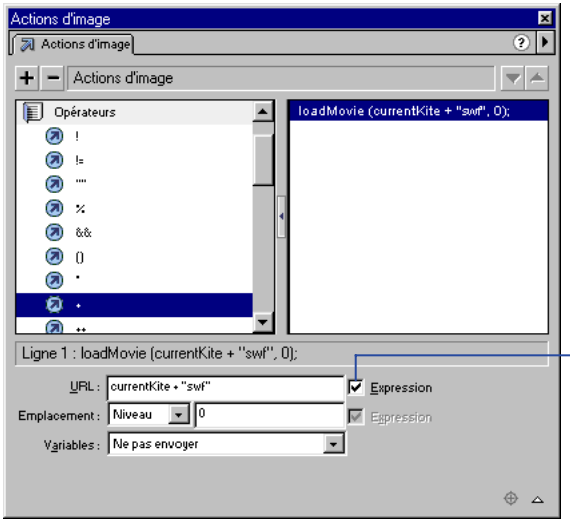
```
function zeroArray (array){
  var i;
  for (i=0; i < array.length; i++) {
    array[i] = 0;
  }
}

var myArray = new Array();
myArray[0] = 1;
myArray[1] = 2;
myArray[2] = 3;

var out = zeroArray(myArray)
```

La fonction `zeroArray` accepte un objet Array comme argument et définit tous les éléments de ce tableau sur 0. Elle peut modifier ce tableau car il est transmis par référence.

Les références aux objets autres qu'aux clips sont appelées *références « en dur »* car si un objet est référencé, il ne peut pas être supprimé. Une référence à un clip est une référence d'un genre spécial appelée *référence « douce »*. Les références douces ne forcent pas l'objet référencé à sortir. Si un clip est détruit par une action du type `removeMovieClip`, les références qui lui sont faites ne fonctionneront plus.

Utilisation des opérateurs pour manipuler les valeurs des expressions

Une expression est une instruction que Flash pourra évaluer et qui renverra une valeur. Vous pouvez créer une expression en combinant des opérateurs et des valeurs ou en appelant une fonction. Lorsque vous écrivez une expression dans le panneau Actions en mode Normal, assurez-vous que la case Expression est cochée dans le panneau Paramètres, sans quoi le champ contiendra la valeur d'une chaîne de caractères.



Les opérateurs sont des caractères qui spécifient comment combiner, comparer ou modifier les valeurs d'une expression. Les éléments sur lesquels les opérateurs agissent sont appelés *opérandes*. Par exemple, dans l'instruction suivante, l'opérateur + ajoute la valeur d'un caractère numérique à la valeur de la variable `foo` ; `foo` et `3` sont des opérandes :

```
foo + 3
```

Cette section décrit les règles générales concernant les types courants d'opérateurs. Pour des informations détaillées sur chaque opérateur mentionné ici, ainsi que sur les opérateurs spécifiques qui n'entrent pas dans cette catégorie, reportez-vous au chapitre 7, « Dictionnaire ActionScript ».

Ordre de priorité des opérateurs

Lorsque deux ou plusieurs opérateurs sont utilisés dans la même instruction, certains opérateurs sont prioritaires sur d'autres. ActionScript suit une hiérarchie précise pour déterminer les opérateurs à exécuter en premier. Par exemple, une multiplication est toujours effectuée avant une addition ; cependant, les éléments entre parenthèses sont prioritaires sur la multiplication. Donc, sans parenthèses, ActionScript effectue la multiplication en premier, comme dans l'exemple suivant :

```
total = 2 + 4 * 3;
```

Le résultat est 14.

Mais si l'addition est mise entre parenthèses, ActionScript effectue l'addition en premier :

```
total = (2 + 4) * 3;
```

Le résultat est 18.

Le tableau de tous les opérateurs et de leur ordre de priorité se trouve en annexe B, « Ordre de priorité et associativité des opérateurs ».

Associativité des opérateurs

Lorsque deux ou plusieurs opérateurs possèdent le même ordre de priorité, leur associativité détermine l'ordre dans lequel ils sont exécutés. L'associativité peut aller de la droite vers la gauche, comme de la gauche vers la droite. Par exemple, l'opérateur de multiplication a une associativité gauche-droite ; ainsi, les deux instructions suivantes sont équivalentes :

```
total = 2 * 3 * 4;
total = (2 * 3) * 4;
```

Le tableau de tous les opérateurs et de leurs associativités se trouve en annexe B, « Ordre de priorité et associativité des opérateurs ».

Opérateurs numériques

Les opérateurs numériques ajoutent, soustraient, multiplient, divisent et effectuent d'autres opérations arithmétiques. Les parenthèses et le signe moins sont des opérateurs arithmétiques. Le tableau suivant répertorie les opérateurs numériques ActionScript :

Opérateur	Opération effectuée
+	Addition
*	Multiplication
/	Division
%	Pourcentage
-	Soustraction
++	Incrémentation
--	Décrémentation

Opérateurs de comparaison

Les opérateurs de comparaison comparent les valeurs des expressions et renvoient une valeur booléenne (`true` ou `false`). Ces opérateurs sont surtout utilisés dans les boucles et les instructions conditionnelles. Dans l'exemple suivant, si la variable `score` est `100`, une animation particulière sera chargée ; sinon, une autre animation sera chargée :

```
if (score == 100){
   loadMovie("winner.swf", 5);
} else {
   loadMovie("loser.swf", 5);
   }
```

Le tableau suivant répertorie les opérateurs de comparaison ActionScript :

Opérateur	Opération effectuée
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à

Opérateurs de chaîne

L'opérateur + agit de manière spéciale sur les chaînes : il concatène les deux opérandes de chaîne. Par exemple, les instructions suivantes :

```
"Congratulations, " to "Donna!":
"Congratulations, " + "Donna!"
```

Le résultat donne `"Congratulations, Donna!"` Si un seul opérande de l'opérateur + est une chaîne, Flash convertit l'autre opérande en chaîne.

Les opérateurs de comparaison, `>`, `>=`, `<` et `<=` agissent également de manière particulière sur les chaînes. Ces opérateurs comparent deux chaînes pour déterminer laquelle apparaît en premier dans l'ordre alphabétique. Les opérateurs de comparaison ne comparent des chaînes que si les deux opérandes sont des chaînes. Si un seul opérande est une chaîne, ActionScript convertit les deux opérandes en nombre et effectue une comparaison numérique.

Remarque : dans Flash 5, la saisie de données ActionScript permet aux opérateurs d'être utilisés dans différents types de données. Il n'est plus nécessaire d'utiliser les opérateurs de chaîne de Flash 4 (par exemple, `eq`, `ge` et `lt`) sauf si vous exportez l'animation vers Flash 4.

Opérateurs logiques

Les opérateurs logiques comparent des valeurs booléennes (`true` et `false`) et renvoient une troisième valeur booléenne. Par exemple, si les deux opérandes sont évalués `true`, l'opérateur logique AND (`&&`) renvoie `true`. Si l'un des opérandes, ou les deux, est évalué `true`, l'opérateur logique OR (`||`) renvoie `false`. Les opérateurs logiques sont souvent utilisés en complément des opérateurs de comparaison pour déterminer la condition d'une action `if`. Par exemple, dans le script suivant, si les deux expressions sont vraies, l'action `if` agira :

```
if ((i > 10) && (_framesloaded > 50)){
   play()
}
```

Le tableau suivant répertorie les opérateurs logiques ActionScript :

Opérateur	Opération effectuée
&&	ET logique
||	OU logique
!	NON logique

Opérateurs au niveau du bit

Les opérateurs au niveau du bit manipulent intérieurement les nombres à virgule flottante pour les transformer en entiers 32 bits, d'emploi plus facile. L'opération exacte au niveau du bit effectuée dépend de l'opérateur, mais toutes les opérations au niveau du bit évaluent chaque chiffre d'un nombre à virgule flottante séparément pour calculer une nouvelle valeur.

Le tableau suivant répertorie les opérateurs au niveau du bit ActionScript :

Opérateur	Opération effectuée
&	Opérateur And au niveau du bit
|	Opérateur Or au niveau du bit
^	Opérateur Xor au niveau du bit
~	Opérateur Not au niveau du bit
«	Décalage gauche
»	Décalage droit
»»	Déplacer le remplissage par zéros vers la droite

Opérateurs d'égalité et d'affectation

Vous pouvez utiliser l'opérateur d'égalité (==) pour déterminer si les valeurs ou les identités de deux opérandes sont équivalentes. Cette comparaison renvoie une valeur booléenne (`true` ou `false`). Si les opérandes sont des chaînes, des nombres ou des valeurs booléennes, ils sont comparés par valeur. Si les opérandes sont des objets ou des tableaux, ils sont comparés par référence.

Vous pouvez utiliser l'opérateur d'affectation (=) pour affecter une valeur à une variable, comme par exemple :

password = "Sk8tEr";

Vous pouvez également utiliser l'opérateur d'affectation pour affecter plusieurs variables dans la même expression. Dans l'instruction suivante, la valeur b est affectée aux variables c et d :

```
a = b = c = d;
```

Vous pouvez aussi utiliser des opérateurs d'affectation composés pour combiner des opérations. Les opérateurs composés agissent sur les deux opérandes et affectent ensuite la nouvelle valeur au premier opérande. Par exemple, les deux instructions suivantes sont équivalentes :

```
x += 15;
x = x + 15;
```

Le tableau suivant répertorie les opérateurs d'égalité et d'affectation ActionScript :

Opérateur	Opération effectuée
==	Égalité
!=	Inégalité
=	Affectation
+=	Addition et affectation
-=	Soustraction et affectation
*=	Multiplication et affectation
%=	Pourcentage et affectation
/=	Division et affectation
<<=	Décalage gauche au niveau du bit et affectation
>>=	Décalage droit au niveau du bit et affectation
>>>=	Déplacer le remplissage par zéros vers la droite et affectation
^=	Opérateur Xor au niveau du bit et affectation
|=	Opérateur Or au niveau du bit et affectation
&=	Opérateur And au niveau du bit et affectation

Opérateurs point et accès tableau

Vous pouvez utiliser les opérateurs point (`.`) et accès tableau (`[]`) pour accéder aux propriétés des objets ActionScript prédéfinies ou personnalisées, y compris celles d'un clip.

L'opérateur point utilise le nom d'un objet dans sa partie gauche et le nom d'une propriété ou d'une variable dans sa partie droite. Le nom de la propriété ou de la variable ne peut pas être une chaîne ni une variable évaluant une chaîne, il doit s'agir d'un identifiant. Les exemples suivants utilisent l'opérateur point :

```
year.month = "Juin";
year.month.day = 9;
```

Les opérateurs point et accès tableau jouent le même rôle, mais l'opérateur point prend un identifiant comme propriété et l'opérateur accès tableau évalue son contenu à un nom et accède ensuite à la valeur de la propriété nommée.
Par exemple, les deux lignes de code suivantes ont accès à la même variable `velocity` dans le clip `rocket` :

```
rocket.velocity;
rocket["velocity"];
```

Vous pouvez utiliser l'opérateur accès tableau pour définir et extraire dynamiquement les noms et les variables des occurrences. Par exemple, dans le code suivant, l'expression insérée dans l'opérateur `[]` est évaluée et le résultat de cette évaluation est utilisé comme nom de la variable qui doit être extraite du clip `name` :

```
name["mc" + i ]
```

Si vous êtes habitué à la syntaxe ActionScript à barre oblique de Flash 4, vous pouvez obtenir la même chose en utilisant la fonction `eval`, par exemple :

```
eval("mc" & i);
```

L'opérateur accès tableau peut également être utilisé dans la partie gauche d'une instruction d'affectation. Cela vous permet de définir dynamiquement les noms d'objets, de variables et d'occurrences, comme dans l'exemple suivant :

```
name[index] = "Gary";
```

Cet exemple est, lui aussi, équivalent à la syntaxe ActionScript à barre oblique de Flash 4 :

```
Set Variable: "name:" & index = "Gary"
```

L'opérateur accès tableau peut également être imbriqué avec lui-même pour simuler des tableaux multidimensionnels.

```
chessboard[row][column]
```

Avec la syntaxe à barre oblique, on obtiendrait l'équivalence suivante :

```
eval("chessboard/" & row & ":" & column)
```

Remarque : si vous souhaitez écrire un script ActionScript compatible avec version 4 de Flash Player, vous pouvez utiliser la fonction `eval` avec l'opérateur `add`.

Utilisation des actions

Les actions sont les instructions, ou les commandes, d'ActionScript. Plusieurs actions affectées à la même image ou au même objet créent un script. Les actions peuvent agir indépendamment les unes des autres, comme dans les instructions suivantes :

```
swapDepths("mc1", "mc2");
gotoAndPlay(15);
```

Vous pouvez également imbriquer les actions en utilisant une action dans une autre, permettant aux actions de s'affecter mutuellement. Dans l'exemple suivant, l'action `if` informe l'action `gotoAndPlay` du moment où elle doit s'effectuer :

```
if (i >= 25) {
   gotoAndPlay(10);
}
```

Les actions peuvent déplacer la tête de lecture du scénario (`gotoAndPlay`), contrôler le déroulement d'un script en créant des boucles (`do while`) ou un opérateur logique conditionnel (`if`) ou créer de nouvelles fonctions et variables (`function`, `setVariable`). Le tableau suivant répertorie toutes les actions ActionScript :

Actions				
break	evaluate	include	print	stopDrag
call	for	loadMovie	printAsBitmap	swapDepths
comment	for...in	loadVariables	removeMovieClip	tellTarget
continue	fsCommand	nextFrame nextScene	return	toggleHighQuality
delete	function	on	setVariable	stopDrag
do...while	getURL	onClipEvent	setProperty	trace
duplicateMovieClip	gotoAndPlay gotoAndStop	play	startDrag	unloadMovie
else	if	prevFrame	stop	var
else if	ifFrameLoaded	prevScene	stopAllSounds	while

Vous trouverez des exemples d'utilisation et de syntaxe pour chaque action en regardant leurs entrées individuelles dans le chapitre 7, « Dictionnaire ActionScript ».

Remarque : dans cet ouvrage, le terme ActionScript *`action`* est synonyme du terme JavaScript *`instruction`*.

Rédaction d'un chemin cible

Pour qu'une action contrôle un clip ou une animation chargée, vous devez spécifier son nom et son adresse, appelés *chemin cible*. Les actions suivantes prennent un ou plusieurs chemins cible comme arguments :

- `loadMovie`
- `loadVariables`
- `unloadMovie`
- `setProperty`
- `startDrag`
- `duplicateMovieClip`
- `removeMovieClip`
- `print`
- `printAsBitmap`
- `tellTarget`

Par exemple, l'action `loadMovie` prend les arguments *`URL`*, *`Location`* et *`Variables`*. *`URL`* indique l'emplacement sur le web de l'animation que vous souhaitez charger. *`Location`* est le chemin cible dans lequel l'animation sera chargée.

```
loadMovie(URL, Location, Variables);
```

Remarque : dans cet exemple, l'argument *`Variables`* n'est pas nécessaire.

L'instruction suivante charge l'URL `http://www.mySite.com/myMovie.swf` dans l'occurrence `bar` du scénario principal root ; `_root.bar` est le chemin cible ;

```
loadMovie("http://www.mySite.com/myMovie.swf", _root.bar);
```

Avec ActionScript, un clip est identifié par son nom d'occurrence. Par exemple, dans l'instruction suivante, la propriété `_alpha` du clip nommé `star` est définie sur 50 % de sa visibilité :

```
star._alpha = 50;
```

Pour donner à un clip un nom d'occurrence :

1. Sélectionnez le clip sur la scène.
2. Choisissez Fenêtre > Panneaux > Occurrence.
3. Entrez un nom d'occurrence dans le champ Nom.

Pour identifier une animation chargée :

Utilisez `_levelX` où `X` est le numéro du niveau spécifié dans l'action `loadMovie` qui a chargé l'animation.

Par exemple, une animation chargée dans le niveau 5 possède le nom d'occurrence `_level5`. Dans l'exemple suivant, une animation est chargée dans le niveau 5 et sa visibilité est définie sur `false` :

```
onClipEvent(load) {
    loadMovie("monClip.swf", 5);
}
onClipEvent(enterFrame) {
   _level5._visible = false;
}
```

Pour entrer le chemin cible d'une animation :

Cliquez sur le bouton Insérer un chemin cible dans le panneau Actions et sélectionnez un clip dans la liste qui s'affiche.

Pour plus d'informations sur la rédaction des chemins cible, voir le chapitre 4, « Utilisation des clips ».

Contrôle du déroulement des scripts

ActionScript utilise les actions `if`, `for`, `while`, `do...while` et `for...in` pour effectuer une action en fonction de l'existence d'une condition.

Utilisation des instructions if

Les instructions qui vérifient si une condition est vraie ou fausse commencent par `if`. Si la condition existe, ActionScript exécute l'instruction qui suit. Si la condition n'existe pas, ActionScript passe à l'instruction suivante, à l'extérieur du bloc de code.

Pour optimiser les performances de votre code, vérifiez d'abord les conditions les plus probables.

Les instructions suivantes testent plusieurs conditions. Le terme `else if` spécifie d'autres tests à effectuer si les conditions précédentes sont fausses.

```
if ((password == null) || (email == null)){
   gotoAndStop("reject");
}  else {
   gotoAndPlay("startMovie");
}
```

Répétition d'une action

ActionScript peut répéter une action un certain nombre de fois précisé ou tant qu'une condition spécifique existe. Utilisez les actions `while`, `do...while`, `for` et `for...in` pour créer des boucles.

Pour répéter une action tant qu'une condition existe :

Utilisez l'instruction `while`.

Une boucle `while` évalue une expression et exécute le code dans le corps de la boucle si l'expression est `true`. L'expression est évaluée après chaque exécution de l'instruction dans la boucle. Dans l'exemple suivant, la boucle est exécutée quatre fois :

```
i = 4
while (i > 0) {
   myMC.duplicateMovieClip("newMC" +i, i );
   i --;
}
```

Vous pouvez utiliser l'instruction `do...while` pour créer le même genre de boucle qu'avec une boucle `while`. Dans une boucle `do...while`, l'expression est évaluée à la fin du bloc de code, donc la boucle est toujours exécutée au moins une fois :

```
i = 4
do {
   myMC.duplicateMovieClip("newMC" +i, i );
   i --;
} while (i > 0);
```

Pour répéter une action en utilisant un compteur intégré :

Utilisez l'instruction `for`.

La plupart des boucles utilisent un compteur d'un certain type pour contrôler le nombre de fois qu'une boucle est exécutée. Vous pouvez déclarer une variable et écrire une instruction qui augmente ou diminue la variable chaque fois que la boucle est exécutée. Dans l'action `for`, le compteur et l'instruction qui l'incrémente font partie de l'action, comme dans l'exemple suivant :

```
for (i = 4; i > 0; i--){
   myMC.duplicateMovieClip("newMC" +i, i + 10);
}
```

Pour itérer sur les enfants d'un clip ou d'un objet :

Utilisez l'instruction `for..in`.

Les enfants sont composés d'autres clips, de fonctions, d'objets et de variables. L'exemple suivant utilise `trace` pour imprimer les résultats dans la fenêtre Résultat :

```
myObject = { name:'Joe', age:25, city:'San Francisco' };
for (propertyName in myObject) {
  trace("myObject has the property: " + propertyName + ", with
the value: " + myObject[propertyName]);
}
```

Cet exemple donne les résultats suivants dans la fenêtre Résultat :

```
myObject has the property: name, with the value: Joe
myObject has the property: age, with the value: 25
myObject has the property: city, with the value: San Francisco
```

Vous pouvez souhaiter que votre script itère sur un type particulier d'enfants, par exemple, seulement sur les enfants du clip. Vous pouvez le faire avec `for...in` en conjonction avec l'opérateur `typeof`.

```
for (name in myMovieClip) {
  if (typeof (myMovieClip[name]) == "movieclip") {
    trace("I have a movie clip child named " + name);
  }
}
```

Remarque : l'instruction `for...in` itère sur les propriétés des objets dans la chaîne prototype de l'objet itéré. Si le prototype d'un objet enfant est `parent`, `for...in` itèrera également sur les propriétés de `parent`. Voir « Création d'un héritage » à la page 84.

Pour plus d'informations sur chaque action, reportez-vous à leurs entrées individuelles dans le chapitre 7, « Dictionnaire ActionScript ».

Utilisation de fonctions prédéfinies

Une fonction est un bloc de code ActionScript qui peut être réutilisé n'importe où dans une animation. Si vous transmettez des valeurs spécifiques, appelées arguments, à une fonction, la fonction agira sur ces valeurs. Une fonction peut également renvoyer des valeurs. Flash possède des fonctions prédéfinies qui vous permettent d'accéder à certaines informations et d'effectuer certaines tâches, comme la détection d'un conflit (`hitTest`), l'obtention de la valeur de la dernière touche enfoncée (`keycode`) et l'obtention du numéro de la version de Flash Player lisant l'animation (`getVersion`).

Appel d'une fonction

Vous pouvez appeler une fonction dans n'importe quel clip de n'importe quel scénario, y compris une animation chargée. Chaque fonction possède ses propres caractéristiques et certaines vous demandent de transmettre des valeurs particulières. Si vous transmettez plus d'arguments qu'il est nécessaire à la fonction, les valeurs supplémentaires sont ignorées. Si vous ne transmettez pas un argument requis, les arguments vides sont affectés du type de données `undefined`, ce qui peut provoquer des erreurs lors de l'exportation du script. Pour appeler une fonction, elle doit se trouver dans l'image que la tête de lecture a atteinte.

Les fonctions prédéfinies de Flash sont énumérées dans le tableau suivant :

Boolean	getTimer	isFinite	newline	scroll
escape	getVersion	isNaN	number	string
eval	globalToLocal	keycode	parseFloat	targetPath
false	hitTest	localToGlobal	parseInt	true
getProperty	int	maxscroll	random	unescape

Remarque : les fonctions de chaîne sont désapprouvées et ne sont pas citées dans le tableau ci-dessus.

Pour appeler une fonction en mode Expert :

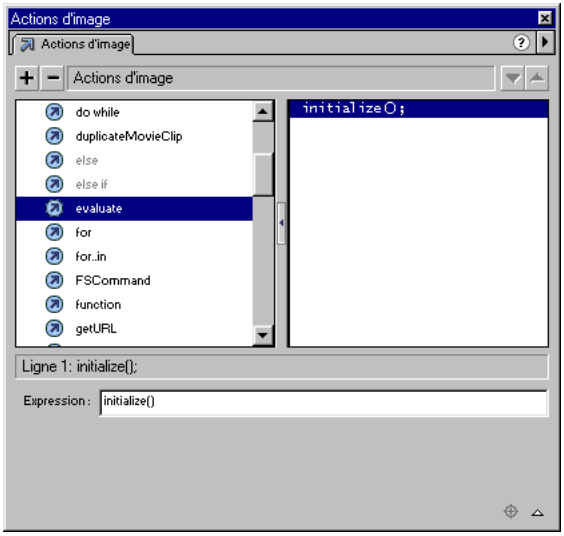
Utilisez le nom de la fonction. Transmettez les arguments requis entre parenthèses.

L'exemple suivant appelle la fonction `initialize` qui ne nécessite aucun argument :

```
initialize();
```

Pour appeler une fonction en mode Normal :

Utilisez l'action `evaluate`. Entrez le nom de la fonction et les arguments requis dans le champ Expression.



Utilisez l'action evaluate pour appeler une fonction en mode Normal.

Pour appeler une fonction d'un autre scénario, utilisez un chemin cible. Par exemple, pour appeler la fonction `calculateTax` déclarée dans l'occurrence `functionsMovieClip`, utilisez le chemin suivant :

```
_root.functionsMovieClip.calculateTax(total);
```

Remarque : transmettez tous les arguments entre parenthèses.

Pour plus d'informations sur chaque fonction, y compris les fonctions de chaîne désapprouvées, reportez-vous à leurs entrées individuelles dans le chapitre 7, « Dictionnaire ActionScript ».

Création de fonctions personnalisées

Vous pouvez définir des fonctions pour exécuter une série d'instructions sur des valeurs transmises. Vos fonctions peuvent également renvoyer des valeurs. Une fois la fonction définie, elle peut être appelée de n'importe quel scénario, y compris le scénario d'une animation chargée.

Une fonction peut être envisagée comme une « boîte noire » : lorsqu'une fonction est appelée, elle est fournie avec une entrée (arguments). Elle effectue des opérations et génère ensuite une sortie (une valeur renvoyée). Une fonction correctement écrite possède des commentaires placés soigneusement concernant son entrée, sa sortie et son but. De cette façon, l'utilisateur de la fonction n'a pas besoin de comprendre avec exactitude son fonctionnement.

Définition d'une fonction

Les fonctions, tout comme les variables, sont attachées au clip qui les définit. Lorsqu'une fonction est redéfinie, la nouvelle définition remplace l'ancienne.

Pour définir une fonction, utilisez l'action `function` suivie du nom de la fonction, des arguments qui doivent être transmis à la fonction et des instructions ActionScript qui indiquent ce que la fonction fait.

L'exemple suivant est une fonction nommée `Circle` avec l'argument `radius` :

```
function Circle(radius) {
   this.radius = radius;
   this.area = Math.PI * radius * radius;
}
```

Remarque : le mot-clé `this`, utilisé dans le corps d'une fonction, fait référence au clip auquel la fonction appartient.

Vous pouvez également définir une fonction en créant un *littéral de fonction*. Un littéral de fonction est une fonction sans nom qui est déclarée dans une expression au lieu d'une instruction. Vous pouvez utiliser un littéral de fonction pour définir une fonction, renvoyer sa valeur et l'affecter à une variable dans une expression, comme dans l'exemple suivant :

```
area = (function () {return Math.PI * radius *radius;})(5);
```

Transmission d'arguments à une fonction

Les arguments sont les éléments sur lesquels une fonction exécute son code (dans cet ouvrage, les termes *arguments* et *paramètres* sont interchangeables.) Par exemple, la fonction suivante prend les arguments `initials` et `finalScore`:

```
function fillOutScorecard(initials, finalScore) {
   scorecard.display = initials;
   scorecard.score = finalScore;
}
```

Lorsque la fonction est appelée, les arguments requis doivent lui être transmis. La fonction substitue les valeurs transmises aux arguments de la définition de la fonction. Dans cet exemple, `scorecard` est le nom d'occurrence d'un clip ; `display` et `score` sont des champs de texte d'entrée dans l'occurrence. L'appel de fonction suivant affecte à la variable `display` la valeur `"JEB"` et à la variable `score` la valeur `45000`:

```
fillOutScorecard("JEB", 45000);
```

L'argument `initials` dans la fonction `fillOutScorecard` est similaire à une variable locale ; il existe tant que la fonction est appelée et cesse d'exister lorsque la fonction sort. Si vous oubliez des arguments lors de l'appel d'une fonction, les arguments oubliés seront transmis comme `undefined`. Si vous fournissez des arguments supplémentaires dans l'appel d'une fonction et qu'ils ne sont pas requis par la déclaration de la fonction, ils sont ignorés.

Utilisation des variables locales dans une fonction

Les variables locales sont des outils intéressants pour organiser le code et en faciliter la compréhension. Lorsqu'une fonction utilise des variables locales, elle peut cacher ses variables de tous les autres scripts de l'animation ; les variables locales ont une portée limitée au corps de la fonction et sont détruites lorsque la fonction sort. Tous les arguments transmis à une fonction sont également traités comme des variables locales.

Remarque : si vous modifiez des variables globales dans une fonction, utilisez les commentaires de script pour annoter ces modifications.

Renvoie des valeurs d'une fonction

Vous pouvez utiliser l'action `return` pour renvoyer les valeurs des fonctions. L'action `return` stoppe la fonction et la remplace par les valeurs de l'action `return`. Si Flash ne rencontre pas d'action `return` avant la fin d'une fonction, la chaîne renvoyée est vide. Par exemple, la fonction suivante renvoie le carré de l'argument `x` :

```
function sqr(x) {
   return x * x;
}
```

Certaines fonctions effectuent une série de tâches sans renvoyer de valeur. Par exemple, la fonction suivante initialise une série de variables globales :

```
function initialize() {
   boat_x = _root.boat._x;
   boat_y = _root.boat._y;
   car_x = _root.car._x;
   car_y = _root.car._y;
}
```

Appel d'une fonction

Pour invoquer une fonction en utilisant le panneau Actions en mode Normal, vous utilisez l'action `evaluate`. Transmettez les arguments requis entre parenthèses. Vous pouvez appeler une fonction dans n'importe quel clip de n'importe quel scénario, y compris une animation chargée. Par exemple, l'instruction suivante invoque la fonction `sqr` dans le clip `MathLib` du scénario principal, lui transmet l'argument 3 et stocke le résultat dans la variable `temp` :

```
var temp = _root.MathLib.sqr(3);
```

Dans Flash 4, pour simuler l'appel d'une fonction, vous pouviez écrire un script sur une image à la fin de l'animation et l'invoquer en transmettant le nom du libellé de l'image à l'action `call`. Par exemple, si un script ayant initialisé des variables se trouvait sur une image libellée `initialize`, vous l'appeliez de cette manière :

```
call("initialize");
```

Ce genre de script ne correspondait pas à une véritable fonction car aucun argument ne pouvait être accepté et aucune valeur ne pouvait être renvoyée. Bien que l'action `call` fonctionne toujours sous Flash 5, son utilisation n'est pas recommandée.

Utilisation d'objets prédéfinis

Vous pouvez utiliser les objets prédéfinis de Flash pour accéder à certains types d'informations. La plupart des objets prédéfinis possèdent des *méthodes* (fonctions affectées à un objet) que vous pouvez appeler pour renvoyer une valeur ou effectuer une action. Par exemple, l'objet Date renvoie les informations de l'horloge du système et l'objet Son vous permet de contrôler les éléments sonores de votre animation.

Certains objets prédéfinis ont des propriétés dont vous pouvez lire les valeurs. Par exemple, l'objet Key possède des valeurs constantes qui représentent les touches du clavier. Chaque objet possède ses propres caractéristiques et capacités que vous pouvez utiliser dans votre animation.

La liste suivante répertorie les objets prédéfinis de Flash :

- Array
- Boolean
- Color
- Date
- Key
- Math
- MovieClip
- Number
- Object
- Selection
- Sound
- String
- XML
- XMLSocket

Les occurrences des clips sont représentées sous forme d'objets dans ActionScript. Vous pouvez appeler les méthodes prédéfinies des clips comme vous le feriez pour tout autre objet ActionScript.

Pour des informations détaillées sur chaque objet, reportez-vous à leurs entrées dans le chapitre 7, « Dictionnaire ActionScript ».

Création d'un objet

Il y a deux façons de créer un objet : l'opérateur `new` et l'opérateur initialisateur d'objet (`{}`). Vous pouvez utiliser l'opérateur `new` pour créer un objet depuis une classe prédéfinie d'objets ou depuis une classe définie et personnalisée d'objets. Vous pouvez utiliser l'opérateur initialisateur d'objet (`{}`) pour créer un objet du type générique Object.

Pour utiliser l'opérateur `new` afin de créer un objet, vous devez l'associer à une fonction constructeur (une fonction constructeur est simplement une fonction dont le seul but est de créer un certain type d'objet). Les objets prédéfinis d'ActionScript sont essentiellement des fonctions constructeur pré-écrites. Le nouvel objet *instancie,* ou crée, une copie de l'objet et lui affecte toutes les propriétés et les méthodes de cet objet. Le principe est le même que pour un clip qui est déplacé de la Bibliothèque vers la scène dans une animation. Par exemple, les instructions suivantes instancient un objet Date :

```
currentDate = new Date();
```

Vous pouvez accéder aux méthodes de quelques objets prédéfinis sans les instancier. Par exemple, l'instruction suivante appelle la méthode `random` de l'objet Math :

```
Math.random();
```

Chaque objet qui requiert une fonction constructeur possède un élément correspondant dans la boîte à outils du panneau Actions, par exemple, `new Color`, `new Date`, `new String` et ainsi de suite.

Pour créer un objet avec l'opérateur `new` en mode Normal :

1. Choisissez `setVariable`.
2. Entrez le nom d'une variable dans le champ Nom.
3. Entrez `new Object`, `new Color` et ainsi de suite dans le champ Valeur. Entrez tous les arguments requis par la fonction constructeur entre parenthèses.
4. Cochez la case Expression du champ Valeur.

   Si vous ne cochez pas la case Expression, la valeur entière sera une chaîne de caractères.

Dans le code suivant, l'objet `c` est créé depuis la fonction constructeur Color :

```
c = new Color(this);
```

Remarque : un nom d'objet est une variable à laquelle le type de données de l'objet est affecté.

Pour accéder à une méthode en mode Normal :

1. Sélectionnez l'action `evaluate`.
2. Entrez le nom de l'objet dans le champ Expression.
3. Entrez une propriété de l'objet dans le champ Expression.

Pour utiliser l'opérateur initialisateur d'objet ({}) en mode Normal :

1. Sélectionnez l'action `setVariable`.
2. Entrez le nom dans le champ Variable ; c'est le nom du nouvel objet.
3. Entrez les paires de noms et de valeurs des propriétés en les séparant par une virgule dans l'opérateur initialisateur d'objet ({}).

Par exemple, dans l'instruction suivante, les noms des propriétés sont `radius` et `area` et leurs valeurs sont `5` et la valeur d'une expression :

```
myCircle = {radius: 5, area:(pi * radius * radius)};
```

Les parenthèses obligent l'expression à s'évaluer. La valeur renvoyée est la valeur de la variable `area`.

Vous pouvez également imbriquer des initialisateurs d'objet et de tableau, comme dans l'instruction suivante :

```
newObject = {name: "John Smith", projects: ["Flash",
"Dreamweaver"]};
```

Pour des informations détaillées sur chaque objet, reportez-vous à leurs entrées dans le chapitre 7, « Dictionnaire ActionScript ».

Accès aux propriétés des objets

Utilisez l'opérateur point (.) pour accéder aux valeurs des propriétés dans un objet. Le nom de l'objet est situé à gauche du point et le nom de la propriété se trouve à droite. Par exemple, dans l'instruction suivante, `myObject` est l'objet et `name` est la propriété :

```
myObject.name
```

Pour affecter une valeur à une propriété en mode Normal, utilisez l'action `setVariable` :

```
myObject.name = "Allen";
```

Pour changer la valeur d'une propriété, affectez une nouvelle valeur comme montré ci-dessous :

```
myObject.name = "Homer";
```

Vous pouvez également utiliser l'opérateur accès tableau ([]) pour accéder aux propriétés d'un objet. Voir « Opérateurs point et accès tableau » à la page 67.

Appel des méthodes d'un objet

Vous pouvez appeler la méthode d'un objet en utilisant l'opérateur point suivi de la méthode. Par exemple, l'instruction suivante appelle la méthode `setVolume` de l'objet Sound :

```
s = new Sound(this);
s.setVolume(50);
```

Pour appeler la méthode d'un objet prédéfini en mode Normal, utilisez l'action `evaluate`.

Utilisation de l'objet MovieClip

Vous pouvez utiliser les méthodes de l'objet prédéfini MovieClip pour contrôler les occurrences des symboles des clips sur la scène. L'exemple suivant dit à l'occurrence `dateCounter` de s'exécuter :

```
dateCounter.play();
```

Pour des informations détaillées sur l'objet MovieClip, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

Utilisation de l'objet Array

L'objet Array est un objet ActionScript prédéfini communément utilisé, qui stocke ses données en propriétés numérotées et non en propriétés nommées. Le nom d'un élément de tableau est appelé *index*. Cette particularité est très pratique pour stocker ou extraire certains types d'informations comme des listes d'étudiants ou une séquence de mouvements dans un jeu.

Vous pouvez affecter les éléments de l'objet Array comme vous le feriez pour la propriété d'un objet quelconque :

```
move[1] = "a2a4";
move[2] = "h7h5";
move[3] = "b1c3";
...
move[100] = "e3e4";
```

Pour accéder au deuxième élément du tableau, utilisez l'expression `move[2]`.

L'objet Array possède une propriété prédéfinie `length` qui correspond à la valeur du nombre d'éléments dans le tableau. Lorsqu'un élément de l'objet Array est affecté et que l'index de l'élément est un entier positif tel que `index >= length`, `length` est automatiquement mis à jour sous la forme `index + 1`.

Utilisation des objets personnalisés

Vous pouvez créer des objets personnalisés pour organiser les informations de vos scripts afin de faciliter le stockage et l'accès en définissant les propriétés et méthodes de l'objet. Après avoir créé l'objet-maître ou « classe », vous pouvez utiliser ou « instancier » des copies (c'est-à-dire des occurrences) de cet objet dans une animation. Cela vous permet de réutiliser le code et de minimiser la taille du fichier.

Un objet est un type de données complexe contenant aucune ou plusieurs propriétés. Chaque propriété, tout comme une variable, possède un nom et une valeur. Les propriétés sont liées à l'objet et contiennent des valeurs qui peuvent être modifiées ou extraites. Ces valeurs peuvent être de n'importe quel type de données : aîne, nombtre, booléen, objet, clip ou `undefined`. Les propriétés suivantes sont de différents types de données :

```
customer.name = "Annie Dupont"
customer.age = 30
customer.member = true
customer.account.currentRecord = 000609
customer.mcInstanceName._visible = true
```

La propriété d'un objet peut également être un objet. A la ligne 4 de l'exemple précédent, `account` est une propriété de l'objet `customer` et `currentRecord` est une propriété de l'objet `account`. Le type de données de la propriété `currentRecord` est un nombre.

Création d'un objet

Vous pouvez utiliser l'opérateur `new` pour créer un objet d'une fonction constructeur Une fonction constructeur possède toujours le même nom que le type d'objet qu'elle crée. Par exemple, une fonction constructeur créant un objet compte sera appelée `Account`. L'instruction suivante crée un nouvel objet avec la fonction appelée `MyConstructorFunction` :

```
new MyConstructorFunction (argument1, argument2, ... argumentN);
```

Lorsque `MyConstructorFunction` est appelée, Flash lui transmet l'argument caché `this`, qui fait référence à l'objet que `MyConstructorFunction` est en train de créer. Lorsque vous définissez une fonction constructeur, `this` vous permet de faire référence aux objets que la fonction créera. Dans l'exemple suivant, la fonction constructeur crée un cercle :

```
function Circle(radius) {
   this.radius = radius;
   this.area = Math.PI * radius * radius;
}
```

Les fonctions constructeur sont souvent utilisées pour remplir les méthodes d'un objet.

```
function Area() {
   this.circleArea = MAth.PI * radius * radius;
}
```

Pour utiliser un objet dans un script, vous devez lui affecter une variable. Pour créer un nouvel objet cercle de rayon 5, utilisez l'opérateur `new` pour créer l'objet et l'affecter à la variable locale `myCircle` :

```
var myCircle = new Circle(5);
```

Remarque : les objets ont la même portée que la variable à laquelle ils sont affectés. Voir « Portée d'une variable » à la page 59.

Création d'un héritage

Toutes les fonctions ont une propriété `prototype` qui est créée automatiquement lors de leur définition. Lorsque vous utilisez une fonction constructeur pour créer un nouvel objet, toutes les propriétés et méthodes de la propriété `prototype` de la fonction deviennent les propriétés et méthodes de la propriété `__proto__` du nouvel objet. La propriété `prototype` indique les valeurs de propriétés par défaut de l'objet créé avec cette fonction. Le fait de transmettre les valeurs utilisant les propriétés `__proto__` et `prototype` s'appelle héritage.

L'héritage fonctionne selon une hiérarchie précise. Lorsque vous appelez la méthode ou la propriété d'un objet, ActionScript vérifie dans l'objet si un tel élément existe. S'il n'existe pas, ActionScript regarde dans la propriété `__proto__` de l'objet pour y chercher les informations (`object.__proto__`). Si la propriété appelée n'est pas une propriété de l'objet `__proto__`, ActionScript cherche dans `object.__proto__.__proto__`.

Il est d'usage courant d'attacher les méthodes à un objet en les affectant à la propriété `prototype` de l'objet. Les étapes suivantes décrivent la manière de définir une méthode échantillon :

1. Définissez la fonction constructeur `Circle` ainsi :

   ```
   function Circle(radius) {
       this.radius = radius;
   }
   ```

2. Définissez la méthode `area` de l'objet `Circle`. La méthode `area` calculera l'aire du cercle. Vous pouvez utiliser un littéral de fonction pour préciser la méthode `area` et définir la propriété area de l'objet prototype circle comme suit :

   ```
   Circle.prototype.area = function () {
          return Math.PI * this.radius * this.radius
       }
   ```

3 Créez une occurrence de l'objet `Circle` comme suit :

```
var myCircle = new Circle(4);
```

4 Appelez la méthode `area` du nouvel objet `myCircle` comme suit :

```
var myCircleArea = myCircle.area()
```

ActionScript cherche l'objet `myCircle` pour la méthode `area`. Puisque l'objet ne possède pas de méthode `area`, son objet prototype `Circle.prototype` est recherché pour la méthode `area`. ActionScript la trouve et l'appelle.

Vous pouvez également attacher une méthode à un objet en attachant la méthode à chaque occurrence individuelle de l'objet, comme dans l'exemple suivant :

```
function Circle(radius) {
   this.radius = radius;
   this.area = function() {
      return Math.PI * this.radius * this.radius
   }
}
```

Cette technique n'est pas recommandée. L'utilisation de l'objet `prototype` est plus efficace car une seule définition d'`area` est nécessaire et cette définition est automatiquement copiée dans toutes les occurrences créées par la fonction `Circle`.

La propriété `prototype` est acceptée par la version 5 de Flash Player et les suivantes. Pour plus d'informations, voir le chapitre 7, « Dictionnaire ActionScript ».

Ouverture des fichiers Flash 4

ActionScript a considérablement changé avec la parution de Flash 5. Il s'agit maintenant d'un langage orienté objet comportant plusieurs types de données et la syntaxe à point. Dans Flash 4, ActionScript ne possédait qu'un seul véritable type de données : chaînes. Il utilisait différents types d'opérateurs dans les expressions pour indiquer si la valeur devait être traitée comme une chaîne ou comme un nombre. Dans Flash 5, vous pouvez utiliser un seul jeu d'opérateurs pour tous les types de données.

Lorsque vous utilisez Flash 5 pour ouvrir un fichier créé sous Flash 4, Flash convertit automatiquement les expressions ActionScript pour les rendre compatibles avec la nouvelle syntaxe de Flash 5. Vous apercevrez les conversions des types de données et des opérateurs dans le code ActionScript :

- L'opérateur = de Flash 4 était utilisé pour des égalités numériques. Dans Flash 5, == est l'opérateur d'égalité et = est l'opérateur d'affectation. Tous les opérateurs = des fichiers Flash 4 sont automatiquement convertis en ==.
- Flash effectue automatiquement les conversions pour s'assurer que les opérateurs fonctionneront correctement. À cause de l'introduction de plusieurs types de données, les opérateurs suivants ont une nouvelle signification :

  `+, ==, !=, <>, <, >, >=, <=`
- Dans ActionScript de Flash 4 , ces opérateurs étaient toujours des opérateurs numériques. Dans Flash 5, ils se comportent différemment, selon le type de données des opérandes. Pour éviter toutes différences sémantiques dans les fichiers importés, la fonction `Number` est intégrée autour des opérandes de ces opérateurs (les nombres constants étant bien évidemment des nombres, ils ne sont pas entourés par `Number`).
- Dans Flash 4, la séquence d'échappement `\n` génère le caractère du retour chariot (ASCII 13). Dans Flash 5, pour correspondre à la norme ECMA-262, `\n` génère le caractère de changement de ligne (ASCII 10). Une séquence `\n` dans les fichiers FLA de Flash 4 est automatiquement convertie en `\r`.
- L'opérateur `&` de Flash 4 était utilisé pour les additions de chaînes. Dans Flash 5, `&` est l'opérateur AND au niveau du bit. L'opérateur d'addition de chaînes est maintenant appelé `add`. Tous les opérateurs `&` des fichiers de Flash 4 sont automatiquement convertis en opérateurs `add`.

- Beaucoup de fonctions de Flash 4 ne nécessitaient pas l'usage des parenthèses, par exemple, `Get Timer` , `Set Variable`, `Stop` et `Play`. Pour créer une syntaxe cohérente, la fonction `getTimer` de Flash 5 et toutes les actions nécessitent maintenant l'usage des parenthèses. Ces parenthèses sont automatiquement ajoutées lors de la conversion.
- Lorsque la fonction `getProperty` est exécutée sur un clip qui n'existe pas, elle renvoie la valeur `undefined`, pas 0, dans Flash 5. Et `undefined == 0` est `false` dans ActionScript de Flash 5. Flash résout ce problème lors de la conversion des fichiers Flash 4 en introduisant les fonctions `Number` dans les comparaisons d'égalité. Dans l'exemple suivant, `Number` obligela conversion de `undefined` en 0 si bien que la comparaison fonctionne :

```
getProperty("clip", _width) == 0
Number(getProperty("clip", _width)) == Number(0)
```

Remarque : si vous avez utilisé des mots-clés de Flash 5 comme noms de variables dans les scripts ActionScript de Flash 4, la syntaxe renverra une erreur dans Flash 5. Pour éviter cela, renommez les variables dans tous les emplacements. Voir « Mots-clés » à la page 53.

Utilisation de Flash 5 pour créer un contenu Flash 4

Si vous utilisez Flash 5 pour créer un contenu pourla version 4 de FlashPlayer (en exportant sous le format de Flash 4), vous ne pourrez pas profiter de toutes les nouvelles caractéristiques ActionScript de Flash 5. Cependant, beaucoup de nouvelles caractéristiques ActionScript sont toujours disponibles. ActionScript de Flash 4 ne possède qu'un seul type de données de base utilisé à la fois pour les manipulations de chaînes et de nombres. Lorsque vous développez une animation pour la version 4 de Flash Player, il vous faut utiliser les opérateurs de chaîne désapprouvés situés dans la catégorie Opérateurs de chaîne dans la boîte à outils.

Vous pouvez utiliser les fonctions Flash 5 suivantes lorsque vous exportez sous le format SWF de Flash 4 :

- L'opérateur d'accès tableau et objet (`[]`).
- L'opérateur point (`.`).
- Les opérateurs logiques, d'affectation, de pré-incrémentation et de post-incrémentation/décrémentation.

- L'opérateur pourcentage (%), toutes les méthodes et propriétés de l'objet Math.

  Ces opérateurs et fonctions ne sont pas acceptés à l'origine par la version 4 de Flash Player. Flash 5 devra les exporter sous forme d'approximations. Cela signifie que les résultats ne seront qu'approximatifs. De plus, à cause de l'introduction d'approximations dans le fichier SWF, ces fonctions prendront plus de place dans les fichiers SWF de Flash 4 qu'elles n'en occupent dans les fichiers SWF de Flash 5.

- Les actions `for`, `while`, `do while`, `break` et `continue`.
- Les actions `print` et `printAsBitmap`.

Les fonctions Flash 5 suivantes ne peuvent pas être utilisées dans les animations exportées au format SWF de Flash 4 :

- Fonctions personnalisées
- Prise en charge de XML
- Variables locales
- Objets prédéfinis (sauf Math)
- Actions sur les clips
- Types de données multiples
- `eval` et la syntaxe à point (par exemple, `eval ("_root.movieclip.variable")`)
- `return`
- `new`
- `delete`
- `typeof`
- `for..in`
- `keycode`
- `targetPath`
- `escape`
- `globalToLocal` et `localToGlobal`
- `hitTest`
- `isFinite` et `inNaN`
- `parseFloat` et `parseInt`
- `tunescape`
- `_xmouse` et `_ymouse`
- `_quality`

CHAPITRE 3
Création d'interaction avec ActionScript

Une animation interactive implique la participation de l'utilisateur. En utilisant le clavier, la souris, ou les deux à la fois, l'utilisateur peut sauter vers différents endroits d'une animation, déplacer des objets, entrer des informations, cliquer sur des boutons et exécuter de nombreuses autres opérations interactives.

Vous créez des animations interactives en créant des scripts qui s'exécutent lorsque des événements spécifiques surviennent. Les événements pouvant déclencher un script se produisent lorsque la tête de lecture arrive à une image, lorsqu'un clip est chargé ou déchargé ou lorsque l'utilisateur clique sur un bouton ou appuie sur des touches du clavier. Vous utilisez ActionScript pour créer des scripts qui indiquent à Flash l'action à exécuter lorsque l'événement survient.

Les actions de base suivantes sont des moyens utilisés généralement pour commander la navigation et l'interaction de l'utilisateur dans une animation :

- Lecture et arrêt des animations
- Réglage de la qualité d'affichage d'une animation
- Arrêt de tous les sons
- Saut jusqu'à une image ou une scène
- Saut jusqu'à une autre URL
- Vérification du chargement d'une image
- Chargement et déchargement des animations supplémentaires

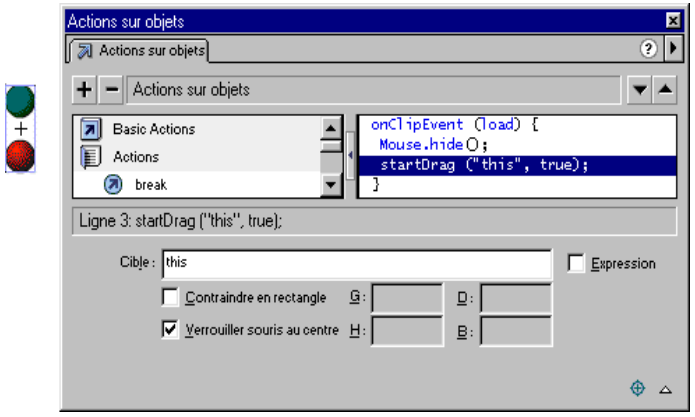
Pour plus d'informations sur ces actions, consultez *Utilisation de Flash*.

Pour créer une interactivité plus complexe, vous devez comprendre les techniques suivantes :

- Création d'un curseur personnalisé
- Obtention de la position de la souris
- Capture des pressions sur les touches
- Création d'un champ de texte déroulant
- Définition des valeurs pour les couleurs
- Création de commandes sonores
- Détection des collisions

Création d'un curseur personnalisé

Pour masquer un curseur standard (c'est-à-dire, la représentation sur l'écran du pointeur de la souris), vous utilisez la méthode `hide` de l'objet prédéfini Souris. Pour utiliser un clip comme curseur personnalisé, vous utilisez l'action `startDrag`.



Actions associées à un clip pour créer un curseur personnalisé.

Pour créer un curseur personnalisé :

1. Créez un clip qui sera utilisé comme curseur personnalisé.
2. Sélectionnez l'occurrence du clip sur la scène.
3. Choisissez Fenêtre > Actions pour ouvrir le panneau Actions sur objets.
4. Dans la liste des boîtes à outils, sélectionnez Objets, puis Souris et faites glisser `hide` jusqu'à la fenêtre Script.

   Le code doit avoir cette forme :

   ```
   onClipEvent(load) {
       Mouse.hide();
   }
   ```

5. Dans la liste des boîtes à outils, sélectionnez Actions et faites glisser `startDrag` jusqu'à la fenêtre Script.
6. Activez la case Verrouiller la souris au centre.

   Le code doit avoir cette forme :

   ```
   onClipEvent(load) {
       Mouse.hide()
       startDrag(this, true);
   }
   ```

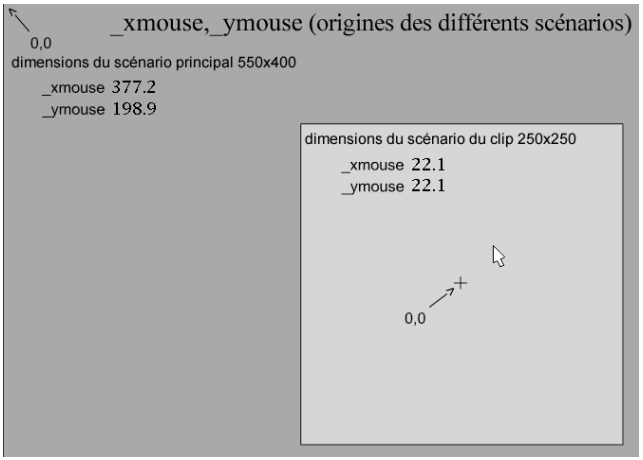
7. Choisissez Contrôle > Tester l'animation pour utiliser le curseur personnalisé.

Les boutons fonctionnent encore lorsque vous utilisez un curseur personnalisé. Il est conseillé de placer le curseur personnalisé sur la couche supérieure du scénario de sorte qu'il se déplace en face des boutons et des autres objets lorsque vous déplacez la souris dans l'animation.

Pour plus informations sur les méthodes de l'objet Souris, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

Obtention de la position de la souris

Vous pouvez utiliser les propriétés `_xmouse` et `_ymouse` pour rechercher la position du pointeur de la souris (curseur) dans une animation. Chaque scénario possède une propriété `_xmouse` et `_ymouse` qui renvoie l'emplacement de la souris basé sur son système de coordonnées.



Les propriétés `_xmouse` et `_ymouse` contenues dans le scénario principal et dans le scénario d'un clip.

Vous pouvez placer l'instruction suivante dans n'importe quel scénario de l'animation `_level0` pour renvoyer la position `_xmouse` à l'intérieur du scénario principal :

```
x_pos = _root._xmouse;
```

Pour déterminer la position de la souris à l'intérieur d'un clip, vous pouvez utiliser le nom de l'occurrence du clip. Par exemple, vous pouvez placer l'instruction suivante dans n'importe quel scénario de l'animation `_level0` pour renvoyer la position `_ymouse` à l'intérieur de l'occurrence `myMovieClip` :

```
y_pos = _root.myMovieClip._ymouse
```

Vous pouvez aussi déterminer la position de la souris à l'intérieur d'un clip en utilisant les propriétés `_xmouse` et `_ymouse` dans une action de clip, de la façon suivante :

```
onClipEvent(enterFrame) {
   xmousePosition = _xmouse;
   xmousePosition = _xmouse;
}
```

Les variables `x_pos` et `y_pos` sont utilisées en tant que conteneurs pour stocker les valeurs des positions de la souris. Vous pouvez utiliser ces variables dans n'importe quel script de votre animation. Dans l'exemple suivant, les valeurs de `x_pos` et `y_pos` sont mises à jour à chaque fois que l'utilisateur déplace la souris.

```
onClipEvent(mouseMove){
   x_pos = _root._xmouse;
   y_pos = _root._ymouse;
}
```

Pour plus informations sur les propriétés `_xmouse` et `_ymouse`, reportez-vous à leurs entrées dans le chapitre 7, « Dictionnaire ActionScript ».

Capture des pressions sur les touches

Vous pouvez utiliser les méthodes de l'objet prédéfini Touche pour détecter la dernière touche enfoncée par l'utilisateur. L'objet Touche ne requiert pas de fonction de construction ; pour utiliser ses méthodes, vous appelez simplement l'objet lui-même, comme dans l'exemple suivant :

```
Key.getCode();
```

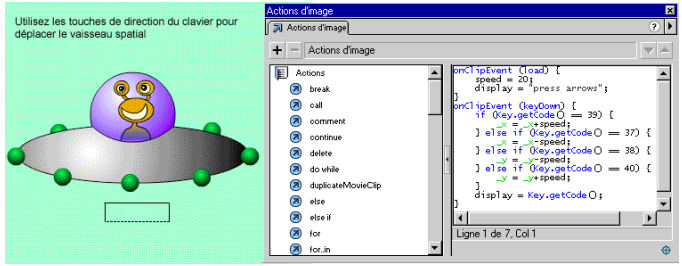
Vous pouvez obtenir soit les codes virtuels de touches soit les valeurs ASCII des pressions sur les touches :

- Pour obtenir le code virtuel de touche de la dernière touche enfoncée par l'utilisateur, utilisez la méthode `getCode`.
- Pour obtenir la valeur ASCII de la dernière touche enfoncée par l'utilisateur, utilisez la méthode `getAscii`.

Un code virtuel de touche est affecté à chaque touche physique du clavier. Par exemple, la touche flèche gauche possède le code virtuel de touche 37. En utilisant un code virtuel de touche, vous pouvez être sûr que les commandes de votre animation sont les mêmes sur chaque clavier, quelle que soit la langue ou la plate-forme utilisée.

Les valeurs ASCII sont affectées aux 127 premiers caractères de chaque jeu de caractères. Les valeurs ASCII fournissent des informations sur un caractère affiché à l'écran. Par exemple, la lettre « A » et la lettre « a » ont des valeurs ASCII différentes.

`Key.getCode` est souvent utilisé dans un gestionnaire `onClipEvent`. Lorsque vous envoyez le paramètre `keyDown`, le gestionnaire indique à ActionScript de vérifier la valeur de la dernière touche enfoncée à condition que l'utilisateur ait réellement appuyé sur une touche. Cet exemple utilise `Key.getCode` dans une instruction `if` pour créer des commandes de navigation pour le vaisseau spatial.



Pour créer des commandes de clavier pour une animation :

1. Choisissez les touches que vous souhaitez utiliser et déterminez leurs codes virtuels de touches en utilisant l'une des approches suivantes :

- Consultez la liste des codes de touches dans l'annexe B, « Touches de clavier et valeurs de code des touches ».
- Utilisez une constante de l'objet Touche (dans la liste des boîtes à outils, sélectionnez Objets, puis Touche. Les constantes apparaissent entièrement en majuscules).
- Affectez l'action de clip suivante, choisissez Contrôle > Tester l'animation et appuyez sur la touche choisie :

```
onClipEvent(keyDown) {
    trace(Key.getCode());
}
```

2. Sélectionnez un clip sur la scène.
3. Choisissez Fenêtre > Actions.
4. Double-cliquez sur l'action `onClipEvent` dans la catégorie Actions de la boîte à outils.
5. Choisissez l'événement `Key down (touche enfoncée)` dans le panneau de configuration.
6. Double-cliquez sur l'action `if` dans la catégorie Actions de la boîte à outils.
7. Cliquez sur le paramètre Condition, sélectionnez Objets, puis Touche et `getCode`.
8. Double-cliquez sur l'opérateur d'égalité (==) dans la catégorie Opérateurs de la boîte à outils.

9 Entrez le code virtuel de touche à droite de l'opérateur d'égalité.

Le code doit avoir cette forme :

```
onClipEvent(keyDown) {
    if (Key.getCode() == 32) {
    }
}
```

10 Sélectionnez l'action à exécuter si l'utilisateur appuie sur la touche appropriée.

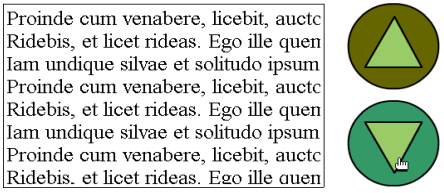
Par exemple, l'action suivante provoque le déplacement du scénario principal vers l'image suivante lorsque l'utilisateur appuie sur la barre d'espace (32) :

```
onClipEvent(keyDown) {
    if (Key.getCode() == 32) {
       nextFrame();
    }
}
```

Pour plus informations sur les méthodes de l'objet Touche, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

Création d'un champ de texte déroulant

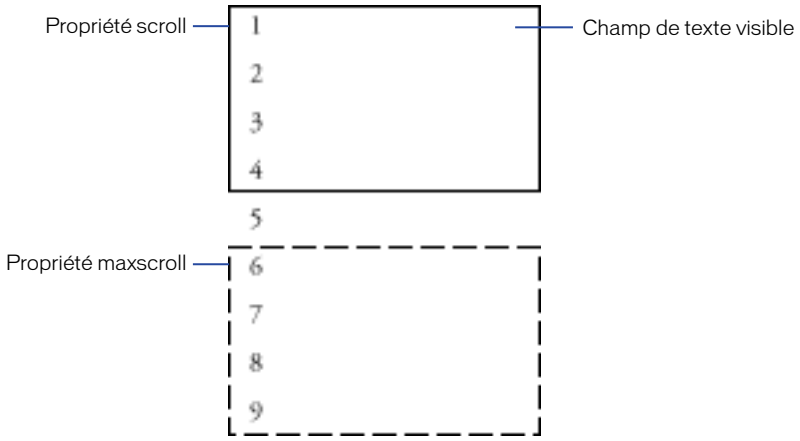
Vous pouvez utiliser les propriétés scroll et maxscroll pour créer un champ de texte déroulant.



Dans le panneau Options de texte, vous pouvez affecter une variable à n'importe quel champ de texte défini sur Texte saisi ou Texte dynamique. Le champ de texte se comporte comme une fenêtre qui affiche la valeur de cette variable.

Chaque variable associée à un champ de texte comporte une propriété `scroll` et une propriété `maxscroll`. Vous pouvez utiliser ces propriétés pour faire défiler du texte dans un champ de texte. La propriété `scroll` renvoie le numéro de la première ligne visible d'un champ de texte ; vous pouvez définir et extraire cette propriété. La propriété `maxscroll` renvoie la première ligne visible d'un champ de texte lorsque la dernière ligne du texte est aussi visible ; vous pouvez lire cette propriété, mais vous ne pouvez pas la définir.

Supposons par exemple que vous ayez un champ de texte de quatre lignes de longueur. Si ce champ contient la variable `speech`, qui remplit neuf lignes du champ de texte, seule une partie de la variable `speech` pourra être affichée à la fois (identifiée par la zone unie) :

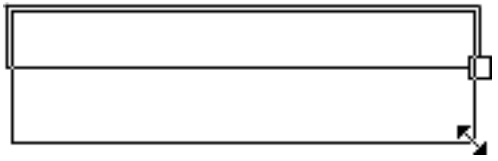


Vous pouvez accéder à ces propriétés en utilisant la syntaxe à point, comme dans l'exemple suivant :

```
textFieldVariable.scroll
myMovieClip.textFieldVariable.scroll
textFieldVariable.maxscroll
myMovieClip.textFieldVariable.maxscroll
```

Pour créer un champ de texte déroulant :

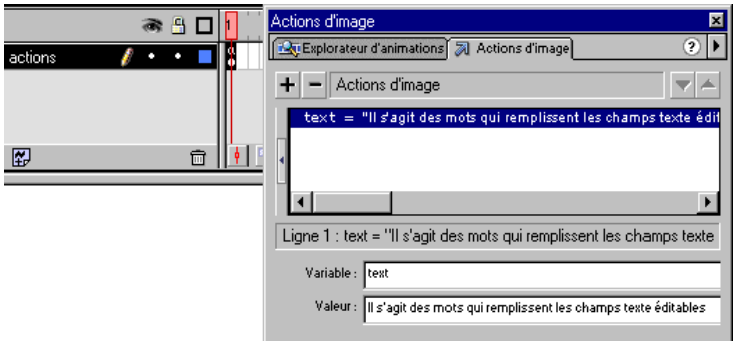
1. Faites glisser un champ de texte sur la scène.
2. Choisissez Fenêtre > Panneaux > Options du texte.
3. Choisissez Texte d'entrée dans le menu contextuel.
4. Entrez le nom de variable **text** dans le champ Variable.
5. Faites glisser l'angle supérieur droit du champ de texte pour le redimensionner.



6 Choisissez Fenêtre > Actions.

7 Sélectionnez l'image 1 dans le scénario principal et affectez une action `set variable` pour définir la valeur de `text`.

Aucun texte n'apparaît dans le champ tant que la variable n'a pas été définie. Par conséquent, même si vous pouvez affecter cette action à n'importe quelle image, bouton ou clip, il est préférable d'affecter l'action à l'image 1 du scénario principal, comme dans l'exemple suivant :



8 Choisissez Fenêtre > Bibliothèques communes > Boutons et faites glisser un bouton jusqu'à la scène.

9 Appuyez sur Alt (Windows) ou sur Option (Macintosh) et faites glisser le bouton pour en créer une copie.

10 Sélectionnez le bouton du haut, puis choisissez Fenêtre > Actions.

11 Faites glisser l'action `set variables` de la boîte à outils jusqu'à la fenêtre Script du panneau Actions.

12 Entrez **text.scroll** dans la zone Variable.

13 Entrez **text.scroll -1** dans la zone Valeur et activez la case à cocher Expression.

14 Sélectionnez le bouton flèche bas et affectez l'action `set variables` suivante :

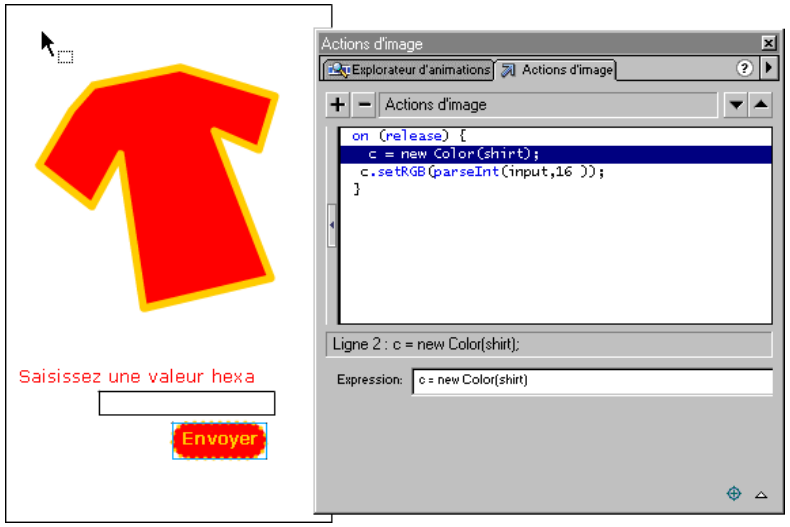
```
text.scroll = text.scroll+1;
```

15 Choisissez Contrôle > Tester l'animation pour tester le champ de texte déroulant.

Pour plus informations sur les propriétés de `scroll` et `maxscroll`, reportez-vous à leurs entrées dans le chapitre 7, « Dictionnaire ActionScript ».

Définition des valeurs pour les couleurs

Vous pouvez utiliser les méthodes de l'objet prédéfini Couleur pour définir la couleur d'un clip. La méthode `setRGB` affecte des valeurs RVB (rouge, vert, bleu) hexadécimales à l'objet, et la méthode `setTransform` définit le pourcentage et les valeurs de décalage pour les composants rouge, vert, bleu et transparent (alpha) d'une couleur. L'exemple suivant utilise `setRGB` pour changer la couleur d'un objet en fonction des valeurs saisies par l'utilisateur.



L'action du bouton crée un objet Couleur et modifie la couleur du tee-shirt en fonction des valeurs saisies par l'utilisateur.

Pour utiliser l'objet Couleur, vous devez créer une occurrence de l'objet et l'appliquer à un clip.

Pour définir la valeur d'une couleur pour un clip :

1. Sélectionnez un clip sur la scène et choisissez Fenêtre > Panneaux > Occurrence.
2. Entrez le nom de l'occurrence **colorTarget** dans la zone Nom.
3. Faites glisser un champ de texte sur la scène.
4. Choisissez Fenêtre > Panneaux > Options de texte et affectez le nom de variable **input**.
5. Faites glisser un bouton jusqu'à la scène et sélectionnez-le.

6 Choisissez Fenêtre > Actions.

7 Faites glisser l'action `set variable` de la boîte à outils jusqu'à la fenêtre Script.

8 Dans la zone Variable, entrez **c**.

9 Dans la boîte à outils, sélectionnez Objets, puis Couleur et faites glisser `new Color` jusqu'à la zone Valeur.

10 Activez la case à cocher Expression.

11 Cliquez sur le bouton Chemin cible et sélectionnez `colorTarget`. Cliquez sur OK.

Le code qui apparaît dans la fenêtre Script doit avoir cette forme :

```
on(release) {
    c = new Color(colorTarget);
}
```

12 Faites glisser l'action `evaluate` de la boîte à outils jusqu'à la fenêtre Script.

13 Dans la zone Expression, entrez **c**.

14 Dans la catégorie Objets de la liste des boîtes à outils, sélectionnez Couleur, puis faites glisser `setRGB` jusqu'à la zone Expression.

15 Sélectionnez Fonctions et faites glisser `parseInt` jusqu'à la zone Expression.

Le code doit avoir cette forme :

```
on(release) {
    c = new Color(colorTarget);
    c.setRGB(parseInt(string, radix));
}
```

16 Pour l'argument de la chaîne `parseInt`, entrez **input**.

La chaîne à analyser correspond à la valeur saisie dans le champ de texte modifiable.

17 Pour l'argument de base `parseInt`, entrez **16**.

La base correspond à la base du système numérique à analyser. Dans ce cas, 16 est la base du système hexadécimal utilisé par l'objet Couleur. Le code doit avoir cette forme :

```
on(release) {
    c = new Color(colorTarget);
    c.setRGB(parseInt(input, 16));
}
```

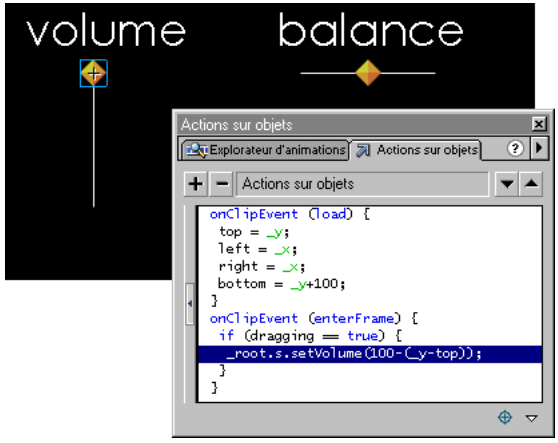
18 Choisissez Contrôle > Tester l'animation pour modifier la couleur du clip.

Pour plus informations sur les méthodes de l'objet Couleur, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

Création de commandes sonores

Pour commander les sons dans une animation, vous utilisez l'objet prédéfini Son. Pour utiliser les méthodes de l'objet Son, vous devez d'abord créer un nouvel objet Son. Vous pouvez utiliser ensuite la méthode `attachSound` pour insérer un son de la bibliothèque dans une animation pendant son exécution.

La méthode `setVolume` de l'objet Son commande le volume, et la méthode `setPan` règle les balances gauche et droite d'un son. L'exemple suivant utilise `setVolume` et `setPan` pour créer des commandes de volume et de balance pouvant être définies par l'utilisateur.



Lorsque l'utilisateur fait glisser le curseur du volume, la méthode `setVolume` est appelée.

Pour associer un son à un scénario :

1. Choisissez Fichier > Importer pour importer un son.
2. Sélectionnez le son dans la bibliothèque et choisissez Liaison dans le menu Options.
3. Sélectionnez Exporter ce symbole et associez-lui l'identifiant **mySound**.
4. Sélectionnez l'image 1 dans le scénario, puis choisissez Fenêtre > Actions.
5. Faites glisser l'action `set variable` de la boîte à outils jusqu'à la fenêtre Script.
6. Dans la zone Valeur, entrez **s**.

7 Dans la liste des boîtes à outils, sélectionnez Objets, puis Son et faites glisser `new Sound` jusqu'à la zone Valeur.

Le code doit avoir cette forme :

```
s = new Sound();
```

8 Double-cliquez sur l'action `evaluate` dans la boîte à outils.

9 Dans la zone Expression, entrez **s**.

10 Dans la catégorie Objets de la liste des boîtes à outils, sélectionnez Son, puis faites glisser `attachSound` jusqu'à la zone Expression.

11 Entrez « **mySound** » dans l'argument ID de `attachSound`.

12 Double-cliquez sur l'action `evaluate` dans la boîte à outils.

13 Dans la zone Expression, entrez **s**.

14 Dans la catégorie Objets, sélectionnez Son, puis faites glisser `start` jusqu'à la zone Expression.

Le code doit avoir cette forme :

```
s = new Sound();
s.attachSound("mySound");
s.start();
```

15 Choisissez Contrôle > Tester l'animation pour écouter le son.

Pour créer une commande de volume réglable :

1 Faites glisser un bouton jusqu'à la scène.

2 Sélectionnez le bouton et choisissez Insertion > Convertir en symbole. Choisissez le comportement du clip.

Ceci crée un clip avec le bouton sur sa première image.

3 Sélectionnez le clip et choisissez Édition > Modifier le symbole.

4 Sélectionnez le bouton et choisissez Fenêtre > Actions.

5 Entrez les actions suivantes :

```
on (press) {
    startDrag ("", false, left, top, right, bottom);
    dragging = true;
}
on (release, releaseOutside) {
    stopDrag ();
    dragging = false;
}
```

Les paramètres de `startDrag`, `left`, `top`, `right` et `bottom` sont des variables définies dans une action de clip.

6 Choisissez Édition > Modifier l'animation pour revenir au scénario principal.

7 Sélectionnez le clip sur la scène.

8 Entrez les actions suivantes :

```
onClipEvent (load) {
    top=_y;
    left=_x;
    right=_x;
    bottom=_y+100;
}

onClipEvent(enterFrame){
    if (dragging==true){
       _root.s.setVolume(100-(_y-top));
    }
}
```

9 Choisissez Contrôle > Tester l'animation pour utiliser la commande de volume réglable.

Pour créer une commande de balance réglable :

1 Faites glisser un bouton jusqu'à la scène.

2 Sélectionnez le bouton et choisissez Insertion > Convertir en symbole. Choisissez la propriété du clip.

3 Sélectionnez le clip et choisissez Édition > Modifier le symbole.

4 Sélectionnez le bouton et choisissez Fenêtre > Actions.

5 Entrez les actions suivantes :

```
on (press) {
    startDrag ("", false, left, top, right, bottom);
    dragging = true;
}
on (release, releaseOutside) {
    stopDrag ();
    dragging = false;
}
```

Les paramètres de `startDrag`, `left`, `top`, `right` et `bottom` sont des variables définies dans une action de clip.

6 Choisissez Édition > Modifier l'animation pour revenir au scénario principal.

7 Sélectionnez le clip sur la scène.

8 Entrez les actions suivantes :

```
onClipEvent(load) {
    top=_y;
    bottom=_y;
    left=_x-50;
    right=_x+50;
    center=_x;
}

onClipEvent(enterFrame){
    if (dragging==true){
       _root.s.setPan((_x-center)*2);
    }
}
```

9 Choisissez Contrôle > Tester l'animation pour utiliser la commande de balance réglable.

Pour plus informations sur les méthodes de l'objet Son, reportez-vous à son entrée dans le chapitre 7, « Dictionnaire ActionScript ».

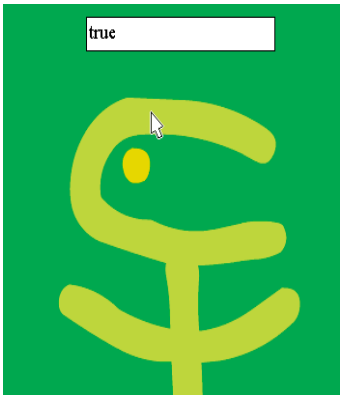
Détection des collisions

Vous pouvez utiliser la méthode `hitTest` de l'objet MovieClip pour détecter les collisions dans une animation. La méthode `hitTest` vérifie si un objet est entré en collision avec un clip et renvoie une valeur booléenne (`true` ou `false`). Vous pouvez utiliser les paramètres de la méthode `hitTest` pour spécifier les coordonnées *x* et *y* d'une zone de correspondance sur la scène, ou utiliser le chemin cible d'un autre clip comme zone de correspondance.

Chaque clip d'une animation est une occurrence de l'objet MovieClip. Ceci vous permet d'appeler des méthodes de l'objet à partir de n'importe quelle occurrence, comme dans l'exemple suivant :

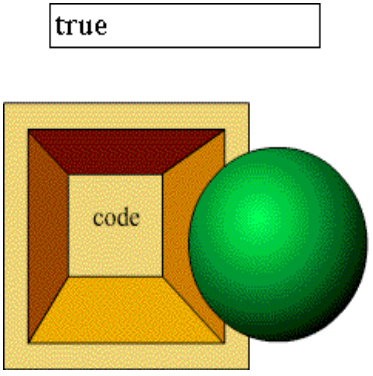
```
myMovieClip.hitTest(target);
```

Vous pouvez utiliser la méthode `hitTest` pour tester la collision entre un clip et un point unique.



« *True* » *apparaît dans le texte rempli chaque fois que le curseur est placé au-dessus de la zone colorée.*

Vous pouvez aussi utiliser la méthode `hitTest` pour tester une collision entre deux clips.



« *True* » *apparaît dans le champ texte chaque fois qu'un clip touche l'autre.*

Pour détecter la collision entre un clip et un point de la scène :

1. Sélectionnez un clip sur la scène.
2. Choisissez Fenêtre > Actions pour ouvrir le panneau Actions sur objets.
3. Double-cliquez sur `trace` dans la catégorie Actions de la boîte à outils.
4. Activez la case à cocher Expression et entrez l'expression suivante dans la zone Expression :

   ```
   trace (this.hitTest(_root._xmouse, _root._ymouse, true);
   ```

   Cet exemple utilise les propriétés `_xmouse` et `_ymouse` comme coordonnées *x* et *y* pour la zone de correspondance et envoie les résultats à la fenêtre de sortie en mode test de l'animation. Vous pouvez aussi définir un champ de texte sur la scène pour afficher les résultats ou utiliser ces derniers dans une instruction `if`.
5. Choisissez Contrôle > Tester l'animation et déplacez la souris sur le clip pour tester la collision.

Pour détecter la collision entre deux clips :

1. Faites glisser deux clips jusqu'à la scène et affectez-leur les noms d'occurrence **mcHitArea** et **mcDrag**.
2. Créez un champ de texte sur la scène et entrez **status** dans la zone Variable d'options de texte.
3. Sélectionnez `mcHitArea` et choisissez Fenêtre > Actions.
4. Double-cliquez sur `evaluate` dans la boîte à outils.
5. Entrez le code suivant dans la zone Expression en sélectionnant des éléments dans la boîte à outils :

   ```
   _root.status=this.hitTest(_root.mcDrag);
   ```
6. Sélectionnez l'action `onClipEvent` dans la fenêtre Script et choisissez l'événement `enterFrame`.
7. Sélectionnez `mcDrag` et choisissez Fenêtre > Actions.
8. Double-cliquez sur `startDrag` dans la boîte à outils.
9. Activez la case à cocher Verrouiller la souris au centre.

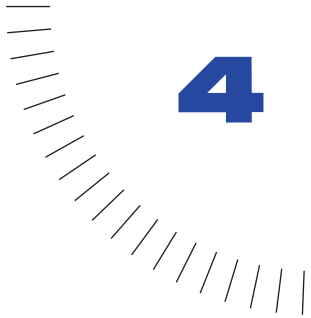
10 Sélectionnez l'action `onClipEvent` dans la fenêtre Script et choisissez l'événement `Mouse down (souris enfoncée)`.

11 Double-cliquez sur `stopDrag` dans la boîte à outils.

12 Sélectionnez l'action `onClipEvent` dans la fenêtre Script et choisissez l'événement `Mouse up (souris relâchée)`.

13 Choisissez Contrôle > Tester l'animation et faites glisser le clip pour tester la détection de collisions.

Pour plus d'informations sur la méthode `hitTest`, reportez-vous à son entrée dans le chapitre 7 « Dictionnaire ActionScript ».



CHAPITRE 4
Utilisation des clips

Un clip est une petite animation Flash : il possède un scénario et des propriétés qui lui sont propres. Un symbole de clip de la bibliothèque peut être utilisé plusieurs fois dans une animation Flash, chaque utilisation étant une *occurrence* du clip. Les clips peuvent s'imbriquer les uns dans les autres. Pour différencier les occurrences, vous affectez à chacune un nom d'occurrence.

Vous pouvez placer toute sorte d'objet dans le scénario d'un clip, y compris d'autres clips. Les animations chargées dans Flash Player en utilisant `loadMovie` sont aussi des petites animations Flash. Chaque clip, chaque animation chargée et le scénario principal d'une animation Flash sont des objets avec des propriétés et des méthodes qui peuvent être manipulés par ActionScript pour créer non seulement des animations complexes, non linéaires, mais aussi une puissante interactivité.

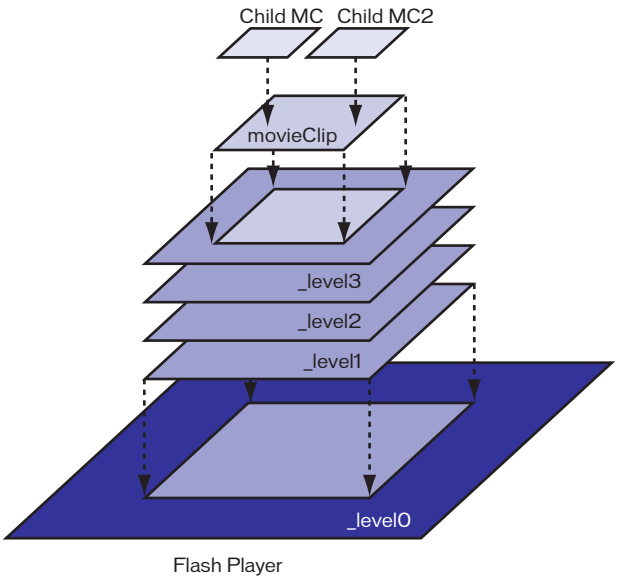
Pour contrôler des clips, vous utilisez des actions et des méthodes de l'objet MovieClip. Vous pouvez associer des actions et des méthodes à des images ou à des boutons dans un clip (actions d'image et de bouton), ou à une occurrence spécifique d'un clip (actions de clip). Les actions d'un clip peuvent contrôler toutes sortes de scénarios dans une animation. Pour contrôler un scénario, vous devez définir son adresse en utilisant un chemin cible. Un chemin cible indique l'emplacement du scénario dans l'animation.

Vous pouvez aussi transformer un clip en smart clip, c'est-à-dire, un clip réalisé grâce à ActionScript que vous pouvez reprogrammer sans utiliser le panneau Actions. Avec les smart clips, un programmeur peut envoyer facilement des objets comportant une logique ActionScript complexe à un concepteur.

À propos des scénarios multiples

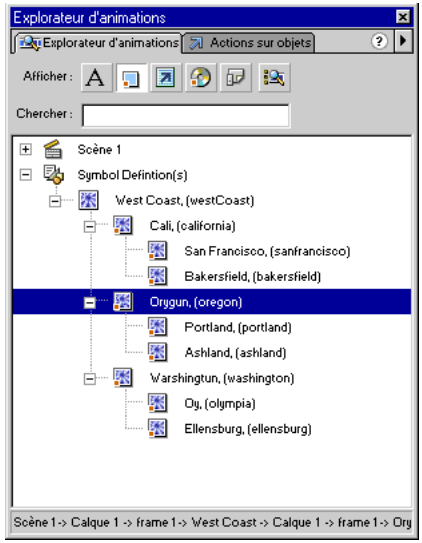
Chaque animation Flash possède un scénario principal situé dans le niveau 0 de Flash Player. Vous pouvez utiliser l'action `loadMovie` pour charger d'autres animations Flash (fichiers SWF) dans Flash Player à n'importe quel niveau au-dessus du niveau 0 (par exemple, niveau 1, niveau 2, niveau 15). Chaque animation chargée dans un niveau de Flash Player inclut un scénario.

Toute animation Flash, quel que soit son niveau, peut disposer d'occurrences de clip dans son scénario. Chaque occurrence de clip peut avoir également un scénario et peut contenir d'autres clips possédant eux aussi des scénarios. Le scénario des clips et les niveaux de Flash Player sont organisés de manière hiérarchique pour vous permettre d'organiser et de contrôler facilement les objets dans votre animation.



La hiérarchie des niveaux et des clips dans Flash Player.

Dans Flash, cette hiérarchie de niveaux et de clips est appelée *liste d'affichages*. Vous pouvez afficher la liste d'affichages dans l'Explorateur d'animations pendant que vous créez un clip ou une animation dans Flash. Vous pouvez afficher la liste d'affichages dans le Débogueur pendant la lecture de l'animation en mode test, dans le lecteur autonome Flash Player ou dans un navigateur web.



L'explorateur d'animations affiche la hiérarchie des scénarios appelée liste d'affichages.

Les scénarios d'une animation Flash sont des objets qui possèdent tous des caractéristiques (propriétés) et des capacités (méthodes) de l'objet prédéfini MovieClip. Les scénarios ont des relations spécifiques entre eux en fonction de leur position dans la liste d'affichages. Les scénarios imbriqués dans d'autres scénarios sont affectés par les modifications apportées au scénario qui les contient. Par exemple, si `portland` est un enfant de `oregon` et que vous modifiez la propriété `_xscale` de `oregon`, la modification s'appliquera également à `portland`.

Les scénarios peuvent aussi échanger des messages entre eux. Par exemple, une action située sur la dernière image d'un clip peut indiquer à un autre clip de s'exécuter.

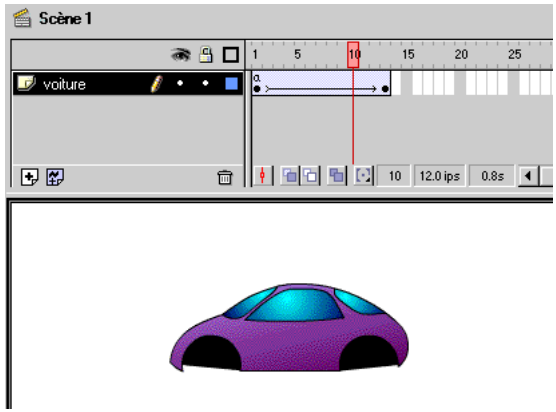
À propos de la relation hiérarchique entre les scénarios

Lorsque vous placez une occurrence de clip dans le scénario d'un autre clip, un symbole de clip contient l'occurrence de l'autre clip. Le premier clip est l'*enfant* et le second le *parent*. Le scénario principal d'une animation est le parent de tous les clips de son niveau.

Les relations parent-enfant qui existent entre les clips sont hiérarchiques. Pour comprendre cette hiérarchie, imaginez la hiérarchie qui existe dans un ordinateur : le disque dur contient un répertoire (ou dossier) principal et des sous-répertoires. Le répertoire principal correspond au scénario principal d'une animation Flash : il est le parent de tout le reste. Les sous-répertoires correspondent aux clips. Vous pouvez utiliser des sous-répertoires pour organiser des données associées.

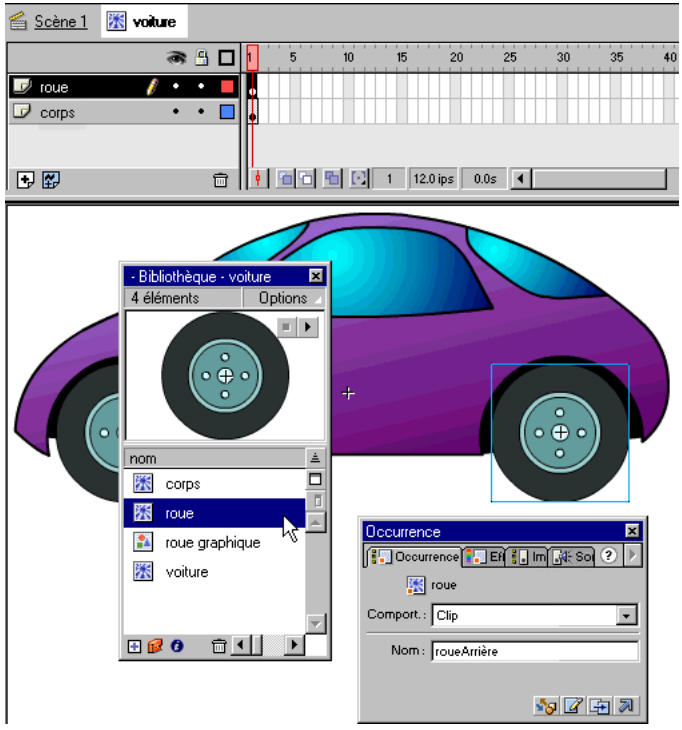
De la même façon, vous pouvez utiliser la hiérarchie des clips dans Flash pour organiser des objets visuels associés comme vous organisez des objets dans le monde réel. Toute modification apportée à un clip parent est aussi appliquée à son enfant.

Par exemple, vous pouvez créer une animation Flash dans laquelle une voiture se déplace à travers la scène. Vous pouvez utiliser un symbole de clip pour représenter la voiture et créer une interpolation de déplacement pour déplacer la voiture à travers la scène.



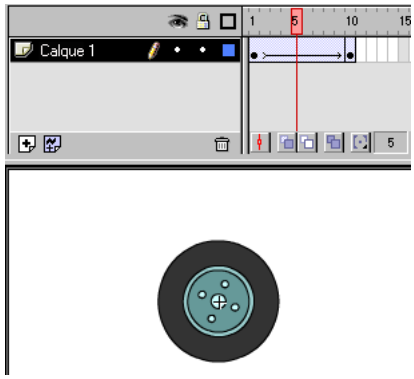
L'interpolation de déplacement déplace le clip de la voiture dans le scénario principal.

La voiture apparaît de côté, avec deux roues visibles. Une fois que la voiture est en mouvement, vous souhaitez ajouter des roues qui tournent. Ainsi, vous créez un clip pour une roue de voiture avec deux occurrences de ce clip, nommées `frontWheel` et `backWheel`. Ensuite, vous placez les roues dans le scénario du clip de la voiture (et non dans le scénario principal). En tant qu'enfants de `car`, `frontWheel` et `backWheel` sont affectés par toute modification apportée à `car`. Cela signifie que les roues se déplaceront avec la voiture lorsque celle-ci traversera la scène.



Les occurrences `roueAvant` et `roueArrière` sont placées dans le scénario du clip `voiture`.

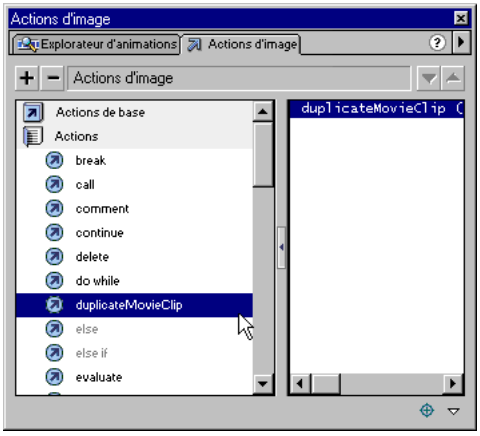
Quant aux roues, vous pouvez définir une interpolation de déplacement qui déclenche la rotation du symbole de la roue pour faire tourner les deux occurrences. Même si vous modifiez `frontWheel` et `backWheel`, ils seront toujours affectés par l'interpolation attribuée à leur clip parent, `car` et les roues tourneront, mais elles se déplaceront avec le clip parent `car` à travers la scène.



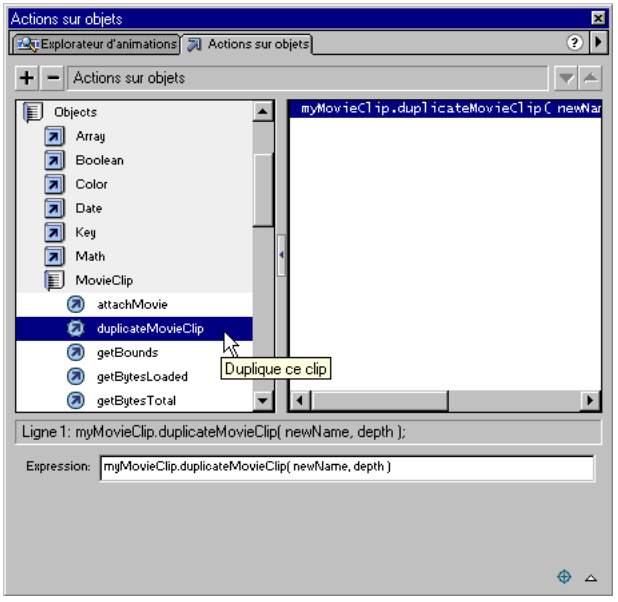
Envoi de messages entre les scénarios

Vous pouvez envoyer des messages entre les scénarios. Le scénario qui contient l'action s'appelle *contrôleur* et celle qui la reçoit s'appelle *cible*. Vous pouvez affecter une action à une image ou un bouton dans un scénario ou, si le scénario correspond à un clip, au clip lui-même.

Pour cibler des scénarios, vous pouvez utiliser des actions de la catégorie Actions, ou vous pouvez utiliser des méthodes de l'objet MovieClip de la catégorie Objets dans le panneau Actions. Par exemple, vous pouvez utiliser l'action `duplicateMovieClip` pour cibler et créer des copies d'occurrences de clip pendant la lecture d'une animation.



Vous pouvez utiliser des actions de la catégorie Actions pour cibler un scénario.



Vous pouvez utiliser des méthodes de l'objet MovieClip pour cibler un scénario.

Pour exécuter plusieurs actions sur la même cible, vous pouvez utiliser l'action `with`. À l'instar de l'instruction JavaScript `with`, l'action `with` d'ActionScript est une enveloppe qui vous permet de définir l'adresse du scénario ciblé une seule fois et d'exécuter une série d'actions sur ce clip. Il n'est pas nécessaire de définir l'adresse du scénario ciblé dans chaque action.

Vous pouvez aussi utiliser l'action `tellTarget` pour exécuter plusieurs actions sur la même cible.

Pour communiquer entre les scénarios, vous devez procéder comme suit :

- Entrez un nom d'occurrence pour le clip cible.

  Pour nommer une occurrence de clip, utilisez le panneau Occurrence (Fenêtre > Panneaux > Occurrence). Les scénarios chargés dans des niveaux utilisent leur numéro de niveau comme nom d'occurrence, par exemple, `_level6`.

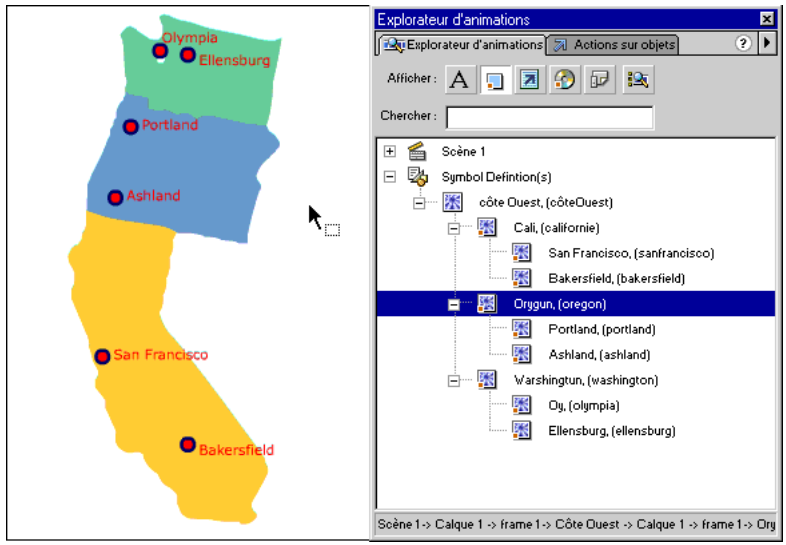
- Entrez le chemin cible du nom d'occurrence dans le panneau Actions.

  Vous pouvez entrer le chemin cible manuellement ou utiliser la boîte de dialogue Insérer un chemin cible pour cibler un clip. Voir « Spécification des chemins cibles » à la page 119.

Remarque : Vous pouvez cibler le scénario d'un clip lors de la lecture, à condition que celui-ci se trouve sur la scène.

À propos des chemins cibles absolu et relatif

Un chemin cible est l'adresse du scénario que vous souhaitez cibler. La liste d'affichages des scénarios dans Flash est similaire à la hiérarchie des fichiers et des dossiers sur un serveur web.



L'explorateur d'animations affiche la liste d'affichages des clips en mode création.

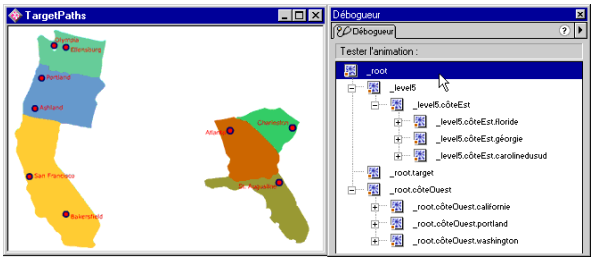
Comme sur un serveur web, vous pouvez définir l'adresse de chaque scénario Flash de deux façons : avec un chemin absolu ou avec un chemin relatif. Le chemin absolu d'une occurrence est toujours le même, quel que soit le scénario qui appelle l'action ; par exemple, le chemin absolu de l'occurrence `california` est toujours `_level0.westCoast.california`. Un chemin relatif varie en fonction de l'endroit à partir duquel il est appelé ; par exemple, le chemin relatif de `california` à partir de `sanfrancisco` est `_parent`, mais à partir de `portland` est `_parent._parent.california`.

Remarque : Pour plus d'informations sur l'Explorateur d'animations, consultez *Utilisation de Flash*.

Un chemin absolu commence par le nom du niveau dans lequel l'animation est chargée et continue à travers la liste d'affichages jusqu'à ce qu'il atteigne l'occurrence cible.

La première animation ouverte dans Flash Player est chargée dans le niveau 0. Vous devez affecter un numéro de niveau à chaque animation chargée par la suite. Le nom cible pour un niveau est `_levelX`, où `X` désigne le numéro de niveau dans lequel vous chargez l'animation. Par exemple, la première animation ouverte dans Flash Player s'appelle `_level0` et une animation chargée dans le niveau 3 s'appelle `_level3`.

Dans l'exemple suivant, deux animations ont été chargées dans le lecteur, `TargetPaths.swf` dans le niveau 0 et `EastCoast.swf` dans le niveau 5. Les niveaux sont indiqués dans le Débogueur, où le niveau 0 apparaît sous la forme `_root`.



Le Débogueur affiche les chemins absolus de tous les scénarios dans la liste d'affichages en mode test de l'animation.

Une occurrence a toujours le même chemin absolu, qu'elle soit appelée à partir d'une action dans une occurrence du même niveau ou à partir d'une action d'un niveau différent. Par exemple, l'occurrence `bakersfield` du niveau 0 a toujours le chemin absolu suivant dans la syntaxe à point :

```
_level0.california.bakersfield
```

Dans la syntaxe à barre oblique, le chemin absolu remplace les points par des barres obliques, comme dans l'exemple suivant :

```
_level0/california/bakersfield
```

Pour communiquer entre les animations de niveaux différents, vous devez utiliser le nom de niveau dans le chemin cible. Par exemple, l'occurrence `portland` définirait l'adresse de l'occurrence `atlanta` comme suit :

```
_level5.georgia.atlanta
```

Dans la syntaxe à point, vous pouvez utiliser l'alias `_root` pour désigner le scénario principal du niveau courant. Pour le scénario principal, ou `_level0`, l'alias _root équivaut à `_level0` lorsqu'il est ciblé par un clip aussi dans `_level0`. Pour une animation chargée dans `_level5`, `_root` équivaut à `_level5` lorsqu'il est ciblé par un clip situé aussi dans le niveau 1. Par exemple, une action appelée à partir de l'occurrence `southcarolina` pourrait utiliser le chemin absolu pour cibler l'occurrence `florida` :

```
_root.eastCoast.florida
```

Dans la syntaxe à barre oblique, vous pouvez utiliser `/` pour désigner le scénario principal du niveau courant comme dans l'exemple suivant :

```
/côteEst/floride
```

Dans la syntaxe à point, que ce soit en mode absolu ou en mode relatif, vous pouvez utiliser les mêmes règles de chemin cible pour identifier une variable dans un scénario ou une propriété d'un objet. Par exemple, l'instruction suivante attribue au nom de variable dans le formulaire d'occurrence la valeur `"Gilbert"` :

```
_root.form.name = "Gilbert";
```

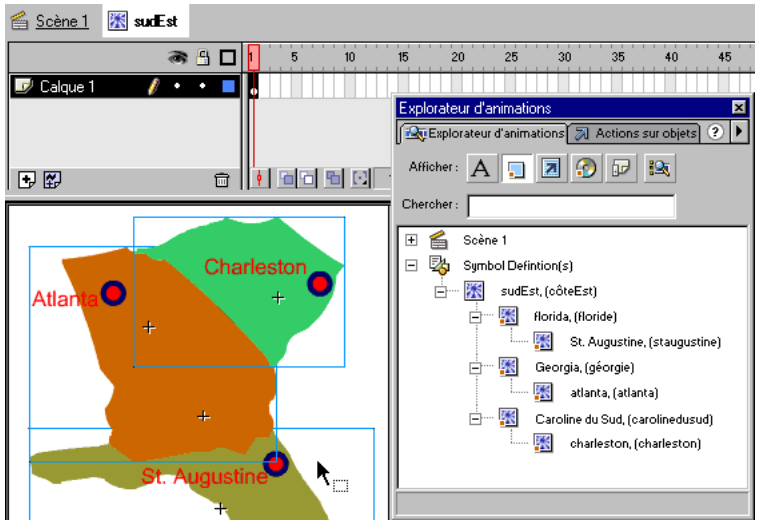
Dans la syntaxe à barre oblique, que ce soit en mode absolu ou en mode relatif, vous pouvez identifier une variable dans le scénario en faisant précéder le nom de variable par deux points (`:`), comme dans l'exemple suivant :

```
/form:name = "Gilbert";
```

Un chemin relatif dépend de la relation qui existe entre le scénario du contrôleur et le scénario cible. Vous pouvez utiliser un chemin relatif pour réutiliser des actions, car la même action peut cibler différents scénarios en fonction de l'endroit où l'action est placée. Les chemins relatifs peuvent définir l'adresse des cibles uniquement à l'intérieur de leur propre niveau dans Flash Player ; ils ne peuvent pas définir l'adresse des animations chargées dans d'autres niveaux. Par exemple, vous ne pouvez pas utiliser un chemin relatif dans une action de `_level0` qui cible un scénario de `_level5`.

Dans la syntaxe à point, vous pouvez utiliser le mot clé `this` dans un chemin cible relatif pour désigner le scénario courant. Vous pouvez utiliser l'alias `_parent` dans un chemin cible relatif pour indiquer le scénario parent du scénario courant. L'alias `_parent` peut être utilisé à plusieurs reprises pour remonter d'un niveau dans la hiérarchie du clip, mais en restant dans le même niveau que Flash Player. Par exemple, `_parent._parent` contrôle un clip situé deux niveaux au-dessus dans la hiérarchie.

Dans l'exemple suivant, chaque ville (`charleston`, `atlanta` et `staugustine`) est un enfant d'une occurrence d'état, et chaque état (`southcarolina`, `georgia` et `florida`) est un enfant de l'occurrence `eastCoast`.



L'Explorateur d'animations illustre les relations parent-enfant qui existent entre les clips.

Une action dans le scénario de l'occurrence `charleston` pourrait utiliser le chemin cible suivant pour cibler l'occurrence `southcarolina` :

```
_parent
```

Pour cibler l'occurrence `eastCoast` à partir d'une action située dans `charleston`, vous pourriez utiliser le chemin relatif suivant :

```
_parent._parent
```

Dans la syntaxe à barre oblique, vous pouvez utiliser deux points (. .) pour remonter d'un niveau dans la hiérarchie. Pour cibler `eastCoast` à partir d'une action située dans `charleston`, vous pourriez utiliser le chemin suivant :

```
../..
```

Pour cibler l'occurrence `atlanta` à partir d'une action située dans le scénario de `charleston`, vous pourriez utiliser le chemin relatif suivant avec la syntaxe à point :

```
_parent._parent.georgia.atlanta
```

Les chemins relatifs sont utiles pour la réutilisation des scripts. Par exemple, vous pouvez associer à un clip un script qui l'agrandit de 150 %, comme suit :

```
onClipEvent (load) {
   _parent._xscale = 150;
   _parent._yscale = 150;
}
```

Vous pouvez ensuite utiliser ce script en le plaçant dans le scénario d'un autre clip.

Pour plus d'informations sur l'adressage et sur la syntaxe à point, consultez « Rédaction de scripts avec ActionScript » à la page 49.

Pour plus d'informations sur la syntaxe à point et la syntaxe à barre oblique, consultez « Utilisation de la syntaxe d'ActionScript » à la page 50.

Spécification des chemins cibles

Pour contrôler un clip ou une animation chargée, vous devez utiliser un chemin cible pour spécifier une cible. Pour cibler un clip, ce dernier doit posséder un nom d'occurrence. Vous pouvez spécifier une cible de différentes façons :

- Entrez un chemin cible en utilisant le bouton et la boîte de dialogue Insérer un chemin cible dans le panneau Actions.
- Entrez manuellement le chemin cible du clip dans votre script.
- Créez une expression en utilisant la référence d'un clip ou les fonctions prédéfinies `targetPath` et `eval`.

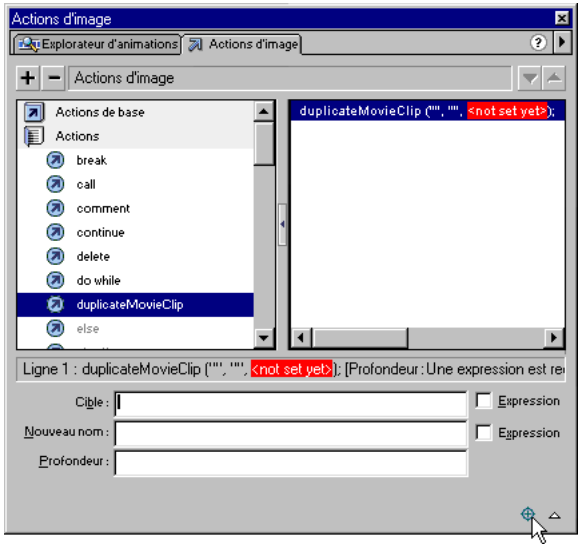
Pour insérer un chemin cible en utilisant la boîte de dialogue Insérer un chemin cible :

1. Sélectionnez l'occurrence de clip, d'image ou de bouton à laquelle vous souhaitez affecter l'action.

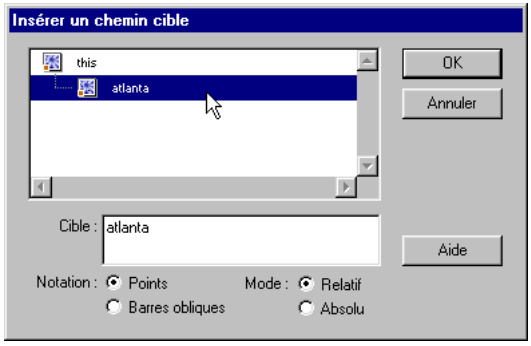
   Celle-ci sera le scénario du contrôleur.

2. Choisissez Fenêtre > Actions pour ouvrir le panneau Actions.
3. Dans la liste des boîtes à outils, choisissez une action dans la catégorie Actions, ou une méthode dans la catégorie MovieClip, à l'intérieur de la catégorie Objets.
4. Cliquez sur le champ Cible ou l'emplacement dans le script pour insérer le chemin cible.

5 Cliquez sur le bouton Insérer un chemin cible, à l'angle inférieur droit du panneau Actions, pour afficher la boîte de dialogue Insérer un chemin cible.



6 Dans la boîte de dialogue Insérer un chemin cible, choisissez une syntaxe : Points (par défaut) ou Barres obliques.



7 Choisissez Absolu ou Relatif comme type de chemin cible.

Voir « À propos des chemins cibles absolu et relatif » à la page 115.

8 Spécifiez votre cible de l'une des façons suivantes :

- Sélectionnez un clip dans la liste d'affichages Insérer un chemin cible.
- Entrez manuellement une cible dans le champ Cible en utilisant un chemin absolu ou relatif et la syntaxe à point.

9 Cliquez sur OK.

Pour insérer un chemin cible manuellement :

Suivez les étapes 1 à 4 ci-dessus et entrez un chemin absolu ou relatif dans le panneau Actions.

Pour utiliser une expression comme chemin cible :

1 Suivez les étapes 1 à 4 ci-dessus.

2 Procédez de l'une des façons suivantes :

- Entrez manuellement une référence comme chemin cible. Une référence est évaluée pour déterminer le chemin cible. Vous pouvez utiliser une référence comme paramètre pour l'action `with`. Dans l'exemple suivant, la variable `index` est évaluée et multipliée par 2. La valeur qui en résulte est utilisée comme nom de clip à l'intérieur de l'occurrence `Bloc` qui doit effectuer la lecture :

```
with (Board.Bloc[index*2]) {
    play();
}
```

- Dans la catégorie Fonctions de la liste des boîtes à outils, choisissez la fonction `targetPath`.

  La fonction `targetPath` convertit la référence d'un clip en une chaîne pouvant être utilisée par des actions telles que `tellTarget`.

  Dans l'exemple suivant, la fonction `targetPath` convertit la référence `Board.Bloc[index*2+1]` en chaîne :

```
tellTarget (targetPath (Board.Bloc[index*2+1])) {
    play();
}
```

  L'exemple précédent est équivalent à l'exemple suivant, qui utilise une syntaxe à barre oblique :

```
tellTarget ("Board/Bloc:" + index*2+1)) {
    play();
}
```

- Dans la catégorie Fonctions de la liste des boîtes à outils, choisissez la fonction `eval`.

  La fonction `eval` convertit une chaîne en référence de clip pouvant être utilisée comme chemin cible par des actions telles que `with`.

  Le script suivant évalue la variable `i`, l'ajoute à la chaîne `"cat"` et affecte la valeur qui en résulte à la variable `x`. La variable `x` est à présent une référence d'une occurrence de clip et peut appeler les méthodes de l'objet MovieClip, comme dans l'exemple suivant :

  ```
  x = eval ("cat" + i)
  x.play()
  ```

  Vous pouvez aussi utiliser la fonction `eval` pour appeler des méthodes directement, comme dans l'exemple suivant :

  ```
  eval ("cat" + i).play()
  ```

Utilisation d'actions et de méthodes pour contrôler des scénarios

Vous pouvez utiliser certaines actions et méthodes de l'objet MovieClip pour *cibler* ou exécuter des tâches sur un clip ou sur un niveau chargé. Par exemple, l'action `setProperty` attribue à une propriété (telle que `_width`) du scénario une valeur (telle que `100`). Certaines méthodes de l'objet MovieClip dupliquent la fonction de toutes les actions qui ciblent les scénarios. Ce sont aussi des méthodes supplémentaires, telles que `hitTest` et `swapDepths`. Que vous utilisiez une action ou une méthode, le scénario cible doit être chargé dans Flash Player lorsque l'action ou la méthode est appelée.

Les actions suivantes peuvent cibler des clips : `loadMovie`, `unloadMovie`, `setProperty`, `startDrag`, `duplicateMovieClip` et `removeMovieClip`. Pour utiliser ces actions, vous devez entrer un chemin cible dans le paramètre Cible de l'action pour indiquer le destinataire de l'action. Certaines de ces actions peuvent cibler des clips ou des niveaux et d'autres ne peuvent cibler que des clips.

Les méthodes de l'objet MovieClip suivantes peuvent contrôler des clips ou des niveaux chargés et n'ont pas d'actions équivalentes : `attachMovie`, `getBounds`, `getBytesLoaded`, `getBytesTotal`, `globalToLocal`, `localToGlobal`, `hitTest` et `swapDepths`.

Lorsqu'une action et une méthode offrent des fonctions similaires, vous pouvez choisir l'une ou l'autre pour contrôler des clips. Le choix dépend de vos préférences et de votre familiarité avec la rédaction de scripts dans ActionScript.

Pour plus d'informations sur les méthodes de l'objet MovieClip et sur chaque action, consultez le chapitre 7, « Dictionnaire ActionScript » à la page 173.

Comparaison entre les méthodes et les actions

Pour utiliser une méthode, vous l'appelez en utilisant le chemin cible du nom d'occurrence, suivi d'un point, puis du nom de la méthode et des arguments, comme dans les instructions suivantes :

```
myMovieClip.play();
parentClip.childClip.gotoAndPlay(3);
```

Dans la première instruction, la méthode `play` déclenche la lecture de l'occurrence `myMovieClip`. Dans la deuxième instruction, la méthode `gotoAndPlay` envoie la tête de lecture située dans `childClip` (qui est un enfant de l'occurrence `clipParent`) vers l'image 3 et démarre la lecture.

Les actions qui contrôlent un scénario ont un paramètre Cible qui spécifie le chemin cible. Par exemple, dans le script suivant, l'action `startDrag` cible l'occurrence `customCursor (curseur personnalisé)` et lui attribue la capacité de se déplacer :

```
on(press){
   startDrag("customCursor");
}
```

Lorsque vous utilisez une méthode, vous appelez la méthode à la fin du chemin cible. Par exemple, l'instruction suivante exécute la même fonction `startDrag` :

```
customCursor.startDrag();
```

Les instructions rédigées en utilisant des méthodes de l'objet MovieClip sont souvent plus brèves, car elles ne requièrent pas l'action `tellTarget`. L'emploi de l'action `tellTarget` est déconseillé, car elle n'est pas compatible avec la norme ECMA-262.

Par exemple, pour demander au clip `myMovieClip` de démarrer la lecture en utilisant les méthodes de l'objet Clip, vous utiliseriez le code suivant :

```
myMovieClip.play();
```

Le code suivant génère les mêmes résultats en utilisant l'action `tellTarget` :

```
tellTarget ("myMovieClip") {
   play();
}
```

Utilisation de plusieurs méthodes ou actions pour cibler un scénario

Vous pouvez utiliser l'action `with` pour définir l'adresse d'un clip ciblé une seule fois et exécuter ensuite une série d'actions sur ce clip. L'action `with` fonctionne sur tous les objets ActionScript, (par exemple Tableau, Couleur et Son) et non seulement sur les clips. L'action `tellTarget` est similaire à l'action `with`. Toutefois, l'action `tellTarget` n'est pas la plus utilisée, car elle ne fonctionne pas avec tous les objets ActionScript et n'est pas conforme à la spécification ECMA-262.

L'action `with` prend un objet comme paramètre. L'objet que vous spécifiez est ajouté à la fin du chemin cible courant. Toutes les actions imbriquées dans une action `with` sont exécutées à l'intérieur du nouveau chemin cible ou *étendue*. Par exemple, dans le script suivant du scénario principal, l'action `with` est communiquée à l'objet `donut.hole (beignet.trou)` pour modifier les propriétés de `hole` :

```
with (donut.hole) {
   _alpha = 20;
   _xscale = 150;
   _yscale = 150;
}
```

C'est comme si l'instruction située à l'intérieur de l'action with était appelée depuis le scénario de l'occurrence `hole`.

Dans l'exemple suivant, notez l'économie que représente l'emploi de l'action `with` et des méthodes de l'objet MovieClip pour émettre plusieurs instructions :

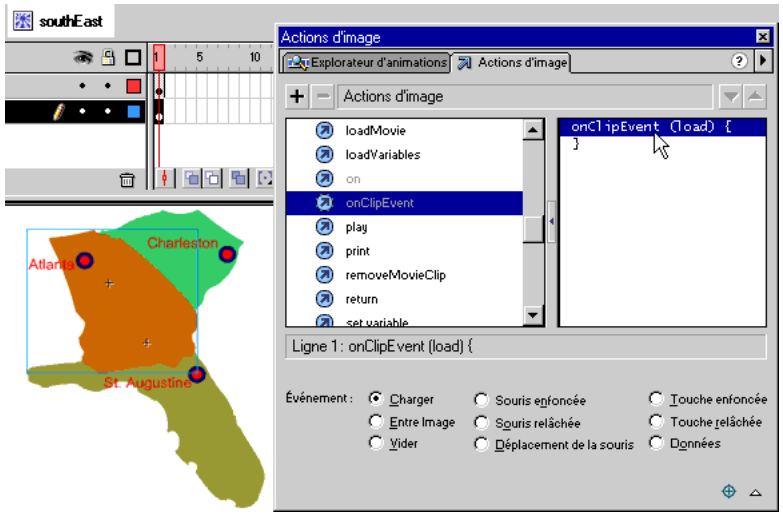
```
with (myMovieClip) {
   _x -= 10;
   _y += 10;
   gotoAndPlay(3);
}
```

Pour plus d'informations sur l'action `tellTarget`, consultez *Utilisation de Flash.*

Affectation d'une action ou d'une méthode

Vous pouvez affecter des actions et des méthodes à un bouton ou une image dans un scénario, ou à une occurrence de clip.

Pour affecter une action ou une méthode à une occurrence de clip, vous devez utiliser un gestionnaire `onClipEvent`. Toutes les actions associées à l'occurrence sont imbriquées dans un gestionnaire `onClipEvent` et s'exécutent une fois que celui-ci a été déclenché. L'action `onClipEvent` est déclenchée par les événements du scénario (tels que le chargement d'une animation) ou par les événements utilisateur (tels que le clic de la souris ou la pression d'une touche). Par exemple, `onClipEvent(mouseMove)` déclenche une action à chaque fois que l'utilisateur déplace la souris.



L'action `onClipEvent` est affectée à une occurrence sur la scène. Les événements `onClipEvent` sont répertoriés dans le panneau de configuration du panneau Actions.

Chargement et déchargement d'animations supplémentaires

Vous pouvez utiliser l'action ou la méthode `loadMovie` pour lire d'autres animations sans fermer Flash Player, ou pour passer d'une animation à une autre sans charger une autre page HTML. Vous pouvez aussi utiliser `loadMovie` pour envoyer des variables à un script CGI, qui génère un fichier SWF en tant que sortie CGI. Lorsque vous chargez une animation, vous pouvez spécifier comme cible un niveau ou un clip, dans lequel sera chargée l'animation.

L'action et la méthode `unloadMovie` supprime une animation chargée précédemment par `loadMovie`. En déchargeant explicitement les animations avec `unloadMovie`, vous assurez une transition en douceur entre les animations et vous pouvez alléger le volume de mémoire requis par Flash Player. Utilisez l'action `loadMovie` pour effectuer les opérations suivantes :

- Lire une séquence d'annonces sous forme de fichiers SWF, en plaçant une action `loadMovie` à la fin de chaque fichier SWF pour charger l'animation suivante.
- Développer une interface de branchement qui permet à l'utilisateur de choisir parmi plusieurs fichiers SWF différents.
- Construire une interface de navigation qui possède des contrôles de navigation dans le niveau 0 qui chargent d'autres niveaux. Le chargement de niveaux garantit une transition plus douce que le chargement de nouvelles pages HTML dans un navigateur.

Modification de la position et de l'apparence d'un clip

Pour modifier les propriétés d'un clip pendant sa lecture, vous pouvez utiliser l'action `setProperty` ou rédiger une instruction qui affecte une valeur à la propriété. Si vous chargez une animation dans une cible, l'animation chargée hérite des propriétés du clip ciblé. Une fois l'animation chargée, vous pouvez modifier les propriétés.

Certaines propriétés, appelées propriétés *en lecture seule*, ont des valeurs que vous pouvez lire mais pas définir. Vous pouvez rédiger des instructions pour définir des propriétés qui ne sont pas des propriétés en lecture seule. L'instruction suivante définit la propriété _alpha de l'occurrence de clip `wheel` qui est un enfant de l'occurrence `car` :

```
car.wheel._alpha = 50;
```

De plus, vous pouvez rédiger des instructions qui extraient la valeur d'une propriété de clip. Par exemple, l'instruction suivante extrait la valeur de la propriété `_xmouse` dans le scénario principal et affecte cette valeur à la propriété `_x` de l'occurrence `customCursor` :

```
onClipEvent(enterFrame) {
   customCursor._x = _root._xmouse;
}
```

Vous pouvez aussi utiliser la fonction `getProperty` pour extraire les propriétés du clip.

Les propriétés `_x`, `_y`, `_rotation`, `_xscale`, `_yscale`, `_height`, `_width`, `_alpha` et `_visible` sont affectées par les transformations effectuées sur le parent du clip et transforment le clip et tous ses enfants. Les propriétés `_rectanglefocus`, `_highquality`, `_quality` et `_soundbuftime` sont globales ; elles appartiennent uniquement au scénario de niveau 0. Toutes les autres propriétés appartiennent à chaque clip ou à chaque niveau chargé. Le tableau suivant répertorie toutes les propriétés de clip :

Propriétés			
_alpha	_highquality	_totalframes	_xscale
_currentframe	_name	_url	_y
_droptarget	_quality	_visible	_ymouse
_focusrect	_rotation	_width	_yscale
_framesloaded	_soundbuftime	_x	
_height	_target	_xmouse	

Déplacement des clips

Vous pouvez utiliser l'action ou la méthode `startDrag` pour attribuer à un clip la capacité de se déplacer pendant la lecture d'une animation. Vous pouvez par exemple créer un clip pouvant être déplacé pour les jeux, les fonctions glisser-déplacer, les interfaces à personnaliser, les barres de défilement et les curseurs de défilement.

Un clip conserve sa capacité de déplacement jusqu'à ce qu'il soit arrêté explicitement par `stopDrag`, ou jusqu'à ce qu'un autre clip soit ciblé avec `startDrag`. Vous pouvez déplacer un seul clip à la fois.

Pour créer des mouvements plus complexes avec les opérations glisser-déplacer, vous pouvez évaluer la propriété `_droptarget` du clip que vous êtes en train de déplacer. Par exemple, vous pouvez examiner la propriété `_droptarget` pour voir si l'animation a été déplacée vers un clip spécifique (tel qu'un clip « poubelle »), puis déclencher une autre action. Consultez « Utilisation des instructions if » à la page 71 et « Utilisation des opérateurs pour manipuler les valeurs des expressions » à la page 62.

Duplication et suppression des clips

Vous pouvez créer ou supprimer des occurrences de clip pendant la lecture de votre animation en utilisant `duplicateMovieClip` ou `removeMovieClip` respectivement. L'action et méthode `duplicateMovieClip` crée dynamiquement une nouvelle occurrence du clip, en lui affectant un nom d'occurrence et une profondeur. Un clip dupliqué commence toujours à l'image 1 même si le clip initial se trouvait dans une autre image lors de la duplication, et il se situe toujours à la tête de tous les clips prédéfinis placés dans le scénario. Les variables ne sont pas copiées dans le clip dupliqué.

Pour supprimer un clip que vous avez créé avec `duplicateMovieClip`, utilisez `removeMovieClip`. Un clip dupliqué est également supprimé si le clip parent est supprimé.

Association de clips

Vous pouvez extraire une copie d'un clip dans une bibliothèque et la lire en tant que partie intégrante de votre animation en utilisant la méthode `attachMovie`. Cette méthode charge un autre clip dans votre clip et le lit pendant l'exécution de l'animation.

Pour utiliser la méthode `attachMovie`, vous devez attribuer au clip associé un nom unique dans la boîte de dialogue Propriétés de liaison de symbole.

Pour nommer un clip pour le partage :

1. Sélectionnez le clip que vous souhaitez associer dans la bibliothèque de l'animation.
2. Dans la fenêtre Bibliothèque, choisissez Liaison dans le menu Options.
3. Pour Liaison, choisissez Exporter ce symbole.
4. Dans la boîte de dialogue Propriétés de liaison de symbole, pour Identifiant, entrez un nom pour le clip. Le nom doit être différent du nom du symbole dans la bibliothèque.
5. Cliquez sur OK.

Pour associer un clip à un autre clip :

1. Dans le panneau Actions, spécifiez la cible à laquelle vous souhaitez associer un clip.
2. Dans la liste des boîtes à outils, sélectionnez l'objet MovieClip, puis la méthode `attachMovie`.

3 Définissez les arguments suivants :

- Pour `idName`, spécifiez le nom d'identifiant que vous avez saisi dans la boîte de dialogue Propriétés de liaison de symbole.
- Pour `newName` (nouveau nom), entrez un nom d'occurrence pour le clip associé afin que vous puissiez le cibler.
- Pour `depth (profondeur)`, entrez le niveau dans lequel l'animation dupliquée sera associée au clip. Chaque animation associée a un ordre d'empilage qui lui est propre ; le niveau 0 étant le niveau de l'animation d'origine. Les clips associés sont toujours au-dessus du clip d'origine.

  Par exemple :

```
myMovieClip.attachMovie("calif", "california", 10 );
```

Création de smart clips

Un smart clip est un clip avec des paramètres de clip définis que vous pouvez modifier. Ces paramètres sont ensuite communiqués à des actions du smart clip, qui modifient le comportement du clip.

Pour créer un smart clip, vous affectez des paramètres de clip à un symbole de clip dans la bibliothèque. Vous pouvez rédiger des instructions ActionScript dans le smart clip et qui s'appliquent aux paramètres du clip, de la même façon que vous utilisez des arguments dans une définition de fonction. Vous pouvez sélectionner une occurrence de smart clip sur la scène et modifier les valeurs des paramètres dans le panneau Paramètres du clip. Au cours de la lecture, les valeurs définies dans le panneau sont envoyées au smart clip avant que les actions de l'animation ne soient exécutées.

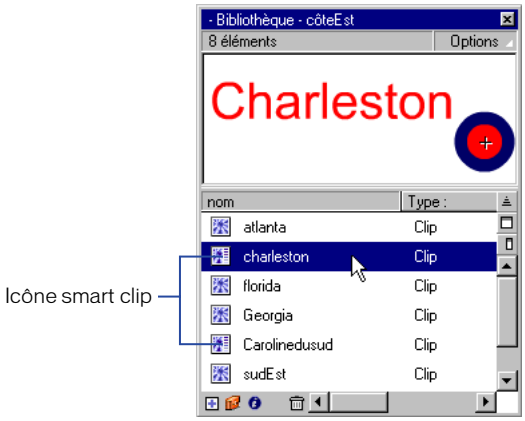
Les smart clips sont utiles pour le programmeur lorsqu'il souhaite communiquer des éléments Flash complexes au concepteur. Le programmeur peut rédiger des actions dans le smart clip avec des variables qui contrôlent le clip et l'animation. Un concepteur peut modifier ensuite les valeurs de ces variables dans le panneau Paramètres du clip sans ouvrir le panneau Actions.

Vous pouvez utiliser des smart clips pour créer des éléments d'interface comme des boutons radio, des menus contextuels, des info-bulles, des enquêtes, des jeux et des avatars. Toute sorte de clip que vous souhaitez réutiliser différemment sans modifier les scripts est adaptée.

Vous pouvez en outre créer une interface personnalisée dans Flash pour le panneau Paramètres du clip afin d'aider les concepteurs qui personnalisent le clip.

Définition des paramètres de clip

Les paramètres de clip sont des données communiquées à un clip lorsqu'il est chargé dans une animation. Vous pouvez définir des paramètres de clip lorsque vous créez votre animation. Vous pouvez utiliser ces paramètres dans des actions pour modifier l'apparence et le comportement du smart clip pendant la lecture de l'animation. Une icône spéciale dans la fenêtre Bibliothèque indique un clip avec des paramètres de clip définis.

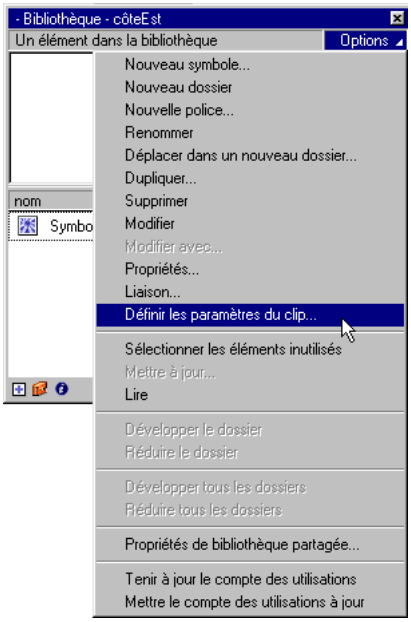


Pour définir des paramètres de clip pour un clip :

1. Sélectionnez un symbole de clip dans la bibliothèque de votre animation et procédez de l'une des façons suivantes pour afficher la boîte de dialogue Paramètres du clip :

- Cliquez avec le bouton droit (Windows) ou cliquez sur Contrôle (Macintosh) et choisissez Définir les paramètres du clip dans le menu contextuel.

- Choisissez Définir les paramètres du clip dans le menu Options, à l'angle supérieur droit de la fenêtre Bibliothèque.



2 Utilisez les commandes de la boîte de dialogue Paramètres du clip comme suit :

- Cliquez sur le bouton Ajouter (+) pour ajouter une nouvelle paire nom/valeur ou des paramètres supplémentaires pour une paire nom/valeur sélectionnée.
- Cliquez sur le bouton Supprimer (-) pour supprimer une paire nom/valeur.
- Utilisez les boutons flèches pour modifier l'ordre des paramètres dans la liste.
- Double-cliquez sur un champ pour le sélectionner, puis entrez une valeur.

3 Pour Nom, entrez un identifiant unique pour le paramètre.

4 Pour Type, choisissez dans le menu contextuel le type de donnée qui sera attribué au paramètre :

- Sélectionnez Valeur par défaut pour utiliser une valeur chaîne ou numérique.
- Sélectionnez Tableau pour obtenir une liste dynamique d'éléments qui peut être agrandie ou rétrécie.
- Sélectionnez Objet pour déclarer plusieurs éléments associés avec des noms et des valeurs, tel qu'un objet point avec des éléments *x* et *y*.
- Sélectionnez Liste pour limiter la sélection à plusieurs choix, tels que `true` ou `false` ou `Red`, `Green` ou `Blue`.

5 Pour Valeur, sélectionnez dans le menu contextuel la valeur par défaut qui sera attribuée au paramètre.

6 Si vous souhaitez utiliser une interface personnalisée pour le panneau Paramètres du clip, procédez de l'une des façons suivantes :

- Entrez un chemin relatif pour le fichier SWF d'interface personnalisée dans le champ Liaison UI personnalisée.
- Cliquez sur le dossier Liaison UI personnalisée et recherchez le fichier SWF d'interface personnalisée.

Voir « Création d'une interface personnalisée » à la page 134.

7 Pour Description, entrez des notes pour décrire chaque paramètre, qui s'afficheront dans le panneau Paramètres du clip.

Dans Description, vous pouvez entrer toute information que vous souhaitez communiquer à une personne utilisant le smart clip. Par exemple, une explication des méthodes que vous avez définies.

8 Choisissez Verrouiller l'occurrence pour éviter que les utilisateurs renomment les paramètres dans le panneau Paramètres du clip.

Il est conseillé de verrouiller les noms de paramètre.

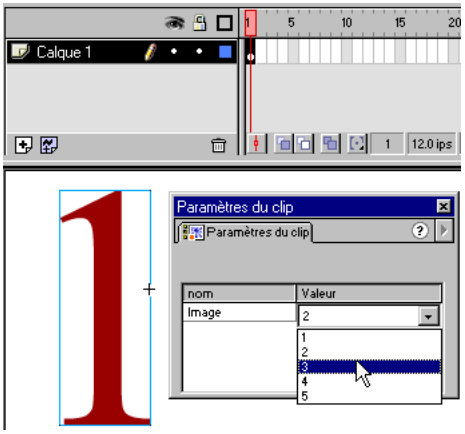
9 Cliquez sur OK.

Définition des paramètres de clip

Dans le smart clip, vous pouvez rédiger des actions qui utilisent les paramètres définis pour modifier le comportement d'un smart clip. Par exemple, si vous définissez un paramètre de clip portant le nom `Frame` (Image), vous pouvez rédiger le script suivant qui utilise le paramètre `Frame` dans le smart clip :

```
onClipEvent(load) {
   gotoAndStop(Image);
}
```

Vous pouvez sélectionner ensuite le smart clip sur la scène et définir la valeur pour le paramètre `Frame` dans le panneau Paramètres du clip afin de changer l'image à lire.



Pour définir les paramètres de clip d'un smart clip :

1. Sélectionnez une occurrence de smart clip sur la scène.

   Les smart clips étant des clips, seule la première image s'affichera en mode création.

2. Choisissez Fenêtre > Panneaux > Paramètres du clip pour afficher le panneau Paramètres du clip.

3. Dans le panneau Paramètres du clip, procédez de l'une des façons suivantes :

- Double-cliquez sur le champ Valeur pour le sélectionner et entrez une valeur pour chaque paramètre.

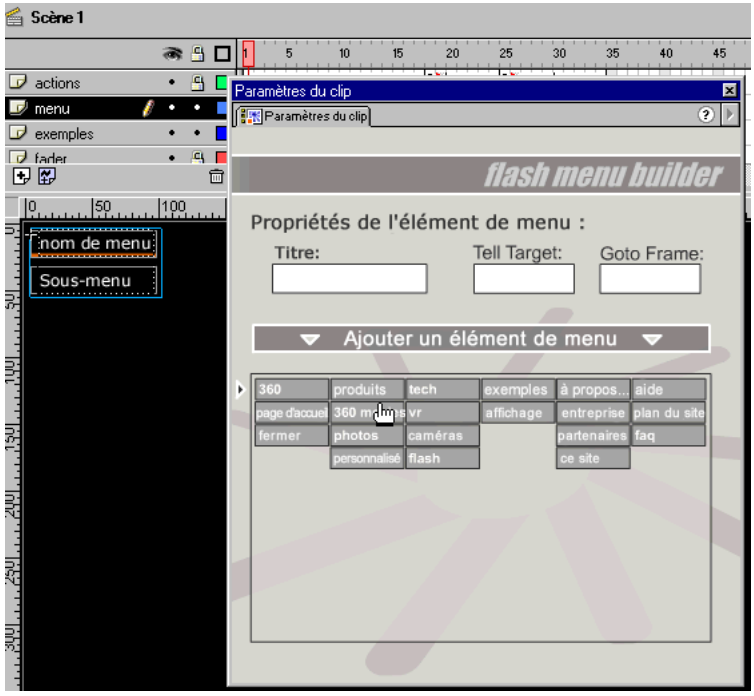
  Si le paramètre a été défini en tant que liste, un menu contextuel s'affiche.

- Si vous avez défini une interface personnalisée, utilisez les éléments d'interface fournis.

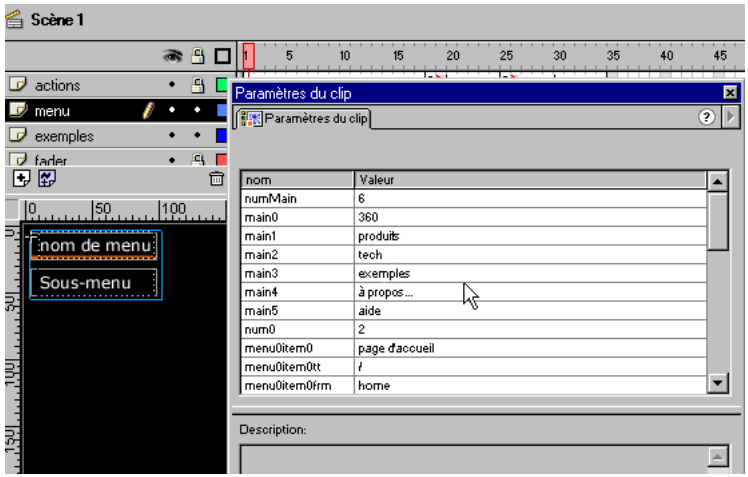
4. Choisissez Contrôle > Tester l'animation pour voir la modification du smart clip.

Création d'une interface personnalisée

Une interface personnalisée est une animation Flash qui vous permet d'entrer des valeurs qui seront communiquées au smart clip. L'interface personnalisée remplace l'interface du panneau Paramètres du clip.



Le panneau Paramètres du clip avec une animation d'interface personnalisée.



Le même smart clip sans interface personnalisée dans le panneau Paramètres du clip.

Toute valeur que vous entrez en utilisant une interface personnalisée est communiquée par le panneau Paramètres du clip au smart clip par le biais d'un intermédiaire ou clip d'échange dans l'interface personnalisée. Le clip d'échange doit avoir comme nom d'occurrence `xch`. Si vous sélectionnez une interface personnalisée dans la boîte de dialogue Définir les paramètres de clip, l'occurrence du smart clip communique les paramètres définis au clip `xch` et toutes les nouvelles valeurs saisies dans l'interface personnalisée sont copiées sur `xch` et renvoyées au smart clip.

Vous devez placer le clip `xch` dans le scénario principal de l'animation d'interface et `xch` doit toujours être chargé. Le clip `xch` doit contenir uniquement les valeurs qui seront communiquées au smart clip. Il ne doit pas contenir des graphiques, d'autres clips ou des instructions ActionScript ; `xch` est un simple conteneur à travers lequel vous envoyez des valeurs. Vous pouvez transférer des objets de niveau supérieur, Tableaux et Objets, par exemple, par le biais du clip `xch`. Cependant, vous ne devez pas envoyer des objets Array ou Object imbriqués.

Pour créer une interface personnalisée pour un smart clip :

1. Choisissez Fichier > Nouveau pour créer une nouvelle animation Flash.
2. Choisissez Insertion > Nouveau symbole pour créer le clip d'échange.
3. Créez un nouveau calque appelé « Clip d'échange »
4. Le calque « Clip d'échange » étant sélectionné, faites glisser le clip d'échange depuis la fenêtre Bibliothèque jusqu'à l'image 1, sur la scène.
5. Sélectionnez le clip d'échange sur la scène, choisissez Fenêtre > Panneaux > Occurrence et entrez le nom `xch`.
6. Créez les éléments d'interface avec lesquels l'auteur va interagir pour définir les paramètres du clip. Par exemple, un menu contextuel, des boutons radio ou des éléments de menu que vous pouvez faire glisser et déplacer.
7. Utilisez l'action `set variable` pour copier des valeurs de variable et d'objet sur l'occurrence `xch`.

   Par exemple, si vous utilisez un bouton comme élément d'interface, ce bouton peut avoir une action qui définit la valeur de la variable `vertical` et la communique à `xch`, comme dans l'exemple suivant :

   ```
   on (release){
       _root.xch.vertical = true;
   }
   ```

8. Exportez l'animation en tant que fichier SWF.

Pour utiliser un fichier SWF d'interface personnalisée avec un smart clip, vous devez les lier dans la boîte de dialogue Définir les paramètres de clip, dans la bibliothèque qui contient le smart clip. Il est conseillé de sauvegarder le fichier SWF dans le même répertoire que le fichier FLA qui contient le smart clip. Si vous réutilisez le smart clip dans un autre fichier ou envoyez le smart clip à un autre développeur, le smart clip et le fichier SWF d'interface personnalisée doivent rester dans les mêmes emplacements relatifs.

CHAPITRE 5
Intégration de Flash dans des applications Internet

Les animations Flash peuvent échanger des informations avec des fichiers distants. Pour envoyer ou charger des variables, vous utilisez les actions `loadVariables` ou `getURL`. Pour charger une animation Flash Player depuis un emplacement distant, vous utilisez l'action `loadMovie`. Pour envoyer ou enregistrer des données XML, vous utilisez l'objet XML ou XMLSocket. Vous pouvez structurer les données XML en utilisant les méthodes prédéfinies de l'objet XML.

Vous pouvez également créer des formulaires Flash composés d'éléments d'interface communs (des champs de texte ou des menus contextuels, par exemple) pour recueillir les données envoyées à une application côté serveur.

Pour améliorer Flash de façon à ce qu'il puisse envoyer ou recevoir les messages provenant de l'environnement hôte de l'animation (par exemple, une fonction Flash Player ou JavaScript dans un navigateur Internet), vous pouvez utiliser `fscommand` et les méthodes Flash Player.

Échange de variables avec un fichier distant

Une animation Flash est une fenêtre permettant de saisir et d'afficher des informations, un peu comme une page HTML. Les animations Flash, contrairement aux pages HTML, peuvent rester chargées dans le navigateur et être mises à jour en permanence avec de nouvelles informations sans qu'il soit nécessaire de les rafraîchir. Vous pouvez utiliser les actions Flash et les méthodes des objets pour envoyer et recevoir des informations dans des scripts, des fichiers texte et des fichiers XML côté serveur.

Les scripts côté serveur peuvent demander des informations particulières à une base de données et les relayer entre la base de données et l'animation Flash. Les scripts côté serveur peuvent être rédigés en plusieurs langages : les plus communs sont Perl, ASP (Microsoft Active Server Pages) et PHP.

Le stockage et l'extraction des informations dans une base de données vous permettent de créer un contenu dynamique et personnalisé pour votre animation. Par exemple, vous pourriez créer un tableau de messages, les profils personnels pour les utilisateurs ou un caddie rappelant ce qu'un utilisateur a acheté afin de déterminer les préférences de l'utilisateur.

Vous pouvez utiliser plusieurs actions et méthodes d'objets ActionScript pour transmettre les informations à ou depuis une animation. Chaque action ou méthode utilise un protocole pour transférer les informations. Chacun nécessite également que les informations soient mises en forme d'une certaine manière.

Les actions suivantes utilisent le protocole HTTP ou HTTPS pour envoyer les informations au format codé URL : `getURL`, `loadVariables`, `loadMovie`.

Les méthodes suivantes utilisent le protocole HTTP ou HTTPS pour envoyer les informations sous forme XML : `XML.send`, `XML.load`, `XML.sendAndLoad`.

Les méthodes suivantes créent et utilisent un support de connexion TCP/IP pour envoyer les informations sous forme XML : `XMLSocket.connect`, `XMLSocket.send`.

À propos de la sécurité

Lorsque vous exécutez une animation Flash dans un navigateur Internet, vous ne pouvez charger les données dans l'animation que depuis un fichier situé sur un serveur du même sous-domaine. Ainsi, les animations Flash n'ont pas la possibilité de télécharger des informations sur les serveurs d'autres personnes.

Pour déterminer le sous-domaine d'une URL constituée d'un ou de deux composants, utilisez le domaine entier :

Domaine	Sous-domaine
http://macromedia	macromedia
http://macromedia.com	macromedia.com

Pour déterminer le sous-domaine d'une URL constituée de plus de deux composants, supprimez le dernier niveau :

Domaine	Sous-domaine
http://x.y.macromedia.com	y.macromedia.com
http://www.macromedia.com	macromedia.com

Le graphique suivant montre comment Flash Player détermine s'il faut ou non autoriser une requête HTTP :

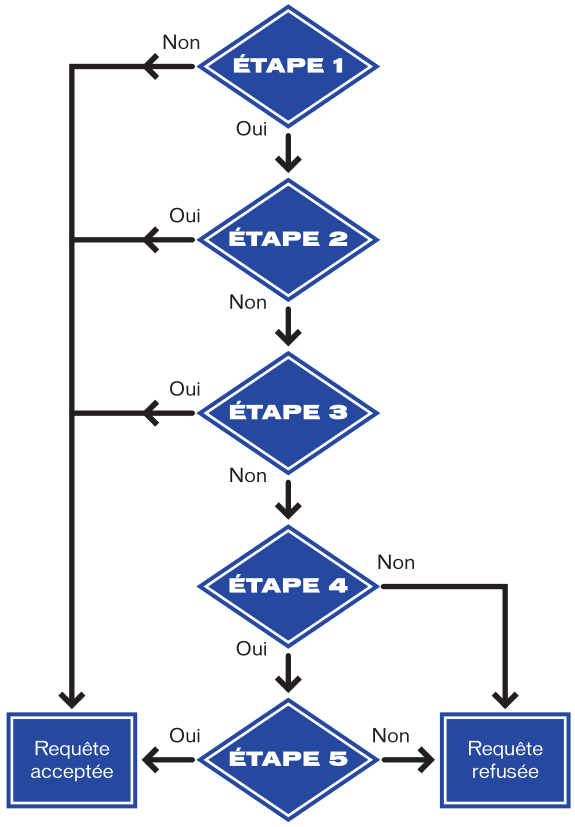
ÉTAPE 1
cette requête est-elle pour : loadVariables, xml.load, xml.sendAndLoad ou xmlsocket.connect ?

ÉTAPE 2
Cette requête est-elle destinée à une URL relative ?

ÉTAPE 3
L'animation à l'origine de la demande est-elle chargée à partir d'un disque local ? (son URL commence par file: ou res:)

ÉTAPE 4
L'URL demandée commence t-elle par http://, https:// ou ftp:// ?

ÉTAPE 5
Le nom de domaine de l'animation à l'origine de la demande correspond-il au nom de domaine de l'URL demandée ?



Lorsque vous utilisez l'objet XMLSocket pour créer un support de connexion avec un serveur, vous devez utiliser un port numéro 1024 ou plus (les ports ayant un numéro inférieur sont généralement utilisés par Telnet, FTP, le World Wide Web ou Finger.)

Flash s'appuie sur les caractéristiques standard de sécurité des navigateurs, ainsi que sur les protocoles HTTP et HTTPS. D'une manière générale, Flash offre la même sécurité que celle disponible avec le standard HTML. Vous devriez suivre les mêmes règles que celles que vous appliquez lorsque vous construisez un site Internet HTML sécurisé. Par exemple, pour prendre en charge des mots de passe sécurisés dans Flash, vous devez définir l'authentification de votre mot de passe en faisant une requête à un serveur Internet.

Pour créer un mot de passe, utilisez un champ de texte pour demander un mot de passe à l'utilisateur. Soumettez-le à un serveur dans une action `loadVariables` ou dans une méthode `XML.sendAndLoad` en utilisant une URL HTTPS avec la méthode `POST`. Le serveur Internet peut alors vérifier si le mot de passe est correct. De cette manière, le mot de passe ne sera jamais disponible dans le fichier SWF.

Vérification des données chargées

Chaque action ou méthode qui charge des données dans une animation (sauf `XMLSocket.send`) est *asynchrone* ; les résultats de l'action sont renvoyés à un moment indéterminé.

Avant de pouvoir utiliser les données chargées dans une animation, vous devez d'abord vérifier si elles ont été chargées. Par exemple, vous ne pouvez pas charger des variables et manipuler les valeurs de ces variables dans le même script. Dans le script suivant, vous ne pouvez pas utiliser la variable `lastFrameVisited` tant que vous n'êtes pas certain que la variable a été chargée depuis le fichier `myData.txt` :

```
loadVariables("myData.txt", 0);
gotoAndPlay(lastFrameVisited);
```

Chaque action et méthode possède une technique spécifique que vous pouvez utiliser pour vérifier les données qui ont été chargées. Si vous utilisez les actions `loadVariables` ou `loadMovie`, vous pouvez charger les informations dans une cible de clip et utiliser l'événement `data` de l'action `onClipEvent` pour exécuter un script. Si vous utilisez l'action `loadVariables` pour charger les données, l'action `onClipEvent(data)` s'exécute lorsque la dernière variable est chargée. Si vous utilisez l'action `loadMovie` pour charger les données, l'action `onClipEvent(data)` s'exécute chaque fois qu'une partie de l'animation est transmise au Flash Player.

Par exemple, l'action du bouton suivante charge les variables depuis le fichier `myData.txt` dans l'animation `loadTargetMC` :

```
on(release) {
   loadVariables("myData.txt", _root.loadTargetMC);
}
```

Une action affectée à l'occurrence `loadTargetMC` utilise la variable `lastFrameVisited` qui est chargée depuis le fichier `myData.txt`. L'action suivante sera exécutée seulement après que toutes les variables, y compris `lastFrameVisited`, aient été chargées :

```
onClipEvent(data){
   gotoAndPlay(lastFrameVisited);
}
```

Si vous utilisez les méthodes `XML.load` et `XMLSocket.connect`, vous pouvez définir un gestionnaire qui traitera les données lorsqu'elles arriveront. Un gestionnaire est une propriété de l'objet XML ou XMLSocket auquel vous affectez une fonction que vous avez définie. Les gestionnaires sont appelés automatiquement lorsque les données sont reçues. Pour l'objet XML, utilisez `XML.onLoad`. Pour l'objet XMLSocket, utilisez `XMLSocket.onConnect`.

Pour plus d'informations, voir « Utilisation de l'objet XML » à la page 144 et « Utilisation de l'objet XMLSocket » à la page 148.

Utilisation des actions loadVariables, getURL et loadMovie

Les actions `loadVariables`, `getURL` et `loadMovie` communiquent toutes avec des scripts côté serveur utilisant le protocole HTTP. Chaque action envoie toutes les variables du scénario auquel elle est attachée et traite sa réponse selon les indications ci-dessous :

- `getURL` renvoie les informations dans une fenêtre de navigateur et non dans Flash Player.
- `loadVariables` charge les variables dans un scénario spécifié dans Flash Player.
- `loadMovie` charge une animation dans un niveau spécifié dans Flash Player.

Lorsque vous utilisez les actions `loadVariables`, `getURL` ou `loadMovie`, vous pouvez spécifier plusieurs arguments :

- *`URL`* est le fichier dans lequel se trouvent les variables distantes.
- *`Location`* est le niveau ou la cible dans l'animation qui reçoit les variables.

  Pour plus d'informations sur les niveaux et les cibles, voir « À propos des scénarios multiples » à la page 108.

  Remarque : L'action `getURL` ne prend pas cet argument.

- *`Variables`* définit la méthode HTTP, `GET` ou `POST`, avec laquelle les variables seront envoyées.

Par exemple, si vous vouliez suivre les meilleurs scores d'un jeu, vous pourriez stocker les scores sur un serveur et utiliser une action `loadVariables` pour les charger dans l'animation chaque fois que quelqu'un joue à ce jeu. L'action pourrait avoir l'aspect suivant :

```
loadVariables("http://www.mySite.com/scripts/high_score.php",
_root.scoreClip, GET)
```

Cet exemple charge les variables du script PHP appelé `high_score.php` dans l'occurrence du clip `scoreClip` en utilisant la méthode HTTP `GET`.

Toutes les variables chargées avec l'action `loadVariables` doivent être au format MIME standard application/x-www-urlformencoded (un format standard utilisé par les scripts CGI). Le fichier que vous spécifiez dans l'argument URL de l'action `loadVariables` doit écrire la variable et les paires de valeurs dans ce format pour que Flash puisse les lire.

Le fichier peut spécifier n'importe quel nombre de variables ; la variable et les paires de valeurs doivent être séparées par l'éperluette (`&`) et les mots à l'intérieur d'une valeur doivent être séparés par le signe plus (+). Par exemple, cette séquence définit plusieurs variables :

```
highScore1=54000&playerName1=rockin+good&highScore2=53455&playerN
ame2=bonehelmet&highScore3=42885&playerName3=soda+pop
```

Pour des informations détaillées sur `loadVariables`, `getURL` et `loadMovie`, reportez-vous à leurs entrées dans le chapitre 7, « Dictionnaire ActionScript ».

À propos du format XML

XML (*Extensible Markup Language*) est en passe de devenir le standard pour l'échange de données structurées dans les applications Internet. Vous pouvez intégrer des données à Flash avec des serveurs qui utilisent la technologie XML pour construire des applications sophistiquées (un service de dialogues en ligne ou un service de courtage, par exemple).

En XML, comme en HTML, vous pouvez utiliser des balises pour *baliser* ou spécifier un corps de texte. En HTML, vous pouvez utiliser des balises prédéfinies pour indiquer comment le texte doit apparaître dans un navigateur Internet (par exemple, la balise `<b>` indique que le texte doit être en gras). En XML, vous définissez des balises qui identifient le type d'une partie de données (par exemple, `<password>VerySecret</password>`). XML fait la distinction entre la structure des informations et la façon dont elles sont affichées. Cela permet d'utiliser et de réutiliser le même document XML dans différents environnements.

Chaque balise XML est appelée un *nœud*, ou un élément. Chaque nœud possède un type (1 – élément XML ou 3 – mode texte) et les éléments peuvent également posséder des attributs. Un nœud imbriqué dans un nœud est appelé *enfant* ou *nœud enfant*. Cette structure hiérarchique des nœuds est appelée DOM (*Document Object Model*) XML, un peu comme le DOM JavaScript, qui correspond à la structure des éléments dans un navigateur Internet.

Dans l'exemple suivant, `<PORTFOLIO>` est le nœud parent ; il ne possède pas d'attributs et contient le nœud enfant `<HOLDING>` qui possède les attributs `SYMBOL`, `QTY`, `PRICE` et `VALUE` :

```
<PORTFOLIO>
   <HOLDING SYMBOL="RICH"
      QTY="75"
      PRICE="245.50"
      VALUE="18412.50" />
</PORTFOLIO>
```

Utilisation de l'objet XML

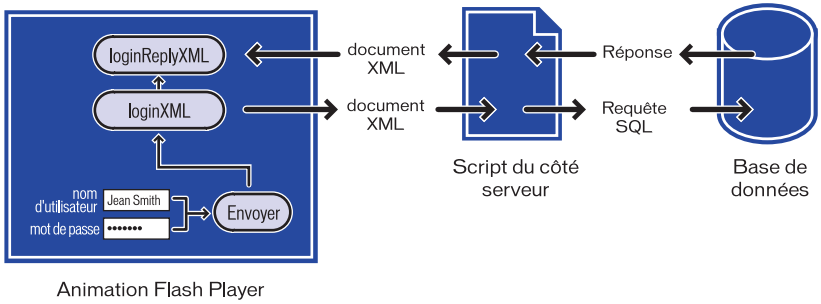
Vous pouvez utiliser les méthodes de l'objet XML ActionScript (par exemple, `appendChild`, `removeNode` et `insertBefore`) pour structurer dans Flash les données XML qui doivent être envoyées à un serveur et pour manipuler et interpréter les données XML téléchargées.

Vous pouvez utiliser les méthodes de l'objet XML pour envoyer et charger des données XML dans un serveur en passant par la méthode HTTP `POST` :

- `load` télécharge le code XML depuis une URL et le place dans un objet XML ActionScript.
- `send` transmet un objet XML à une URL. Toutes les informations en retour sont envoyées dans une fenêtre de navigateur.
- `sendAndLoad` envoie un objet XML à une URL. Toutes les informations en retour sont placées dans un objet XML ActionScript.

Par exemple, vous pourriez créer un système de courtage pour des titres commerciaux qui stockerait toutes ses informations (noms d'utilisateur, mot de passe, ID de session, contenu du portefeuille et informations de transaction) dans une base de données.

Le script côté serveur qui transmet les informations entre Flash et la base de données lit et écrit les données au format XML. Vous pouvez utiliser ActionScript pour convertir les informations récupérées dans l'animation Flash (par exemple, un nom d'utilisateur et un mot de passe) en un objet XML et envoyer ensuite les données au script côté serveur sous forme de document XML. Vous pouvez également utiliser ActionScript pour charger le document XML que le serveur renvoie, sous forme d'objet XML devant être utilisé dans l'animation.



Flux et conversion des données entre une animation Flash Player, un document script côté serveur et une base de données.

La validation du mot de passe pour le système de courtage nécessite deux scripts : une fonction définie sur l'image une et un script qui crée et envoie les objets XML attachés au bouton Envoyer dans le formulaire.

Lorsqu'un utilisateur entre des informations dans les champs de texte de l'animation Flash avec les variables `username` et `password`, les variables doivent être converties en XML avant d'être transmises au serveur. La première section du script charge les variables dans un objet XML nouvellement créé et appelé `loginXML`. Lorsqu'un utilisateur presse sur le bouton Envoyer, l'objet `loginXML` est converti en une chaîne XML et envoyé au serveur.

Le script suivant est attaché au bouton Envoyer. Pour comprendre le script, lisez les commentaires de chaque script précédés par les signes `//` :

```
on (release) {
   // A. Construct a XML document with a LOGIN element
   loginXML = new XML();
   loginElement = loginXML.createElement("LOGIN");
   loginElement.attributes.username = username;
   loginElement.attributes.password = password;
   loginXML.appendChild(loginElement);

   // B. Construct a XML object to hold the server's reply
   new XML();
   loginReplyXML.onLoad = onLoginReply;

   // C. Send the LOGIN element to the server,
   //    place the reply in loginReplyXML
   loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi",
            loginReplyXML);
}
```

La première section du script génère le code XML suivant lorsque l'utilisateur appuie sur le bouton Envoyer :

```
<LOGIN USERNAME="JeanSmith" PASSWORD="VerySecret" />
```

Le serveur reçoit le code XML, génère une réponse XML et la renvoie à l'animation Flash. Si le mot de passe est accepté, le serveur envoie la réponse suivante :

```
<LOGINREPLY STATUS="OK" SESSION="rnr6f7vkj2oe14m7jkkycilb" />
```

Ce code XML comprend un attribut `SESSION` qui contient une ID de session unique générée au hasard et qui sera utilisée dans toutes les communications entre le client et le serveur pour le reste de la session. Si le mot de passe est rejeté, le serveur répond par le message suivant :

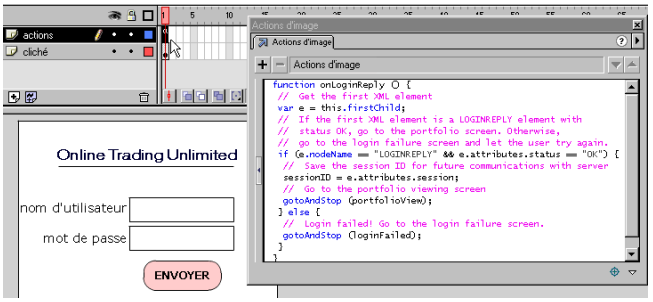
```
<LOGINREPLY STATUS="FAILED" />
```

Le nœud XML `LOGINREPLY` doit charger dans un objet XML vide de l'animation Flash. L'instruction suivante crée l'objet XML `loginreplyXML` pour recevoir le nœud XML :

```
// B. Construct an XML object to hold the server's reply
loginReplyXML = new XML();
loginReplyXML.onLoad = onLoginReply;
```

La seconde instruction affecte la fonction `onLoginReply` au gestionnaire `loginReplyXML.onLoad`.

L'élément XML `LOGINREPLY` arrive de manière asynchrone, un peu comme les données d'une action `loadVariables` et charge dans l'objet `loginReplyXML`. Lorsque les données arrivent, la méthode `onLoad` de l'objet `loginReplyXML` est appelée. Vous devez définir la fonction `onLoginReply` et l'affecter au gestionnaire `loginReplyXML.onLoad` pour qu'il puisse traiter l'élément `LOGINREPLY`. La fonction `onLoginReply` est affectée à l'image qui contient le bouton Envoyer.



La fonction `onLoginReply` est définie dans la première image de l'animation.

La fonction `onLoginReply` est définie dans la première image de l'animation. Pour comprendre le script, lisez les commentaires de chaque script précédés par les signes `//` :

```
function onLoginReply() {
   // Get the first XML element
   var e = this.firstChild;
   // If the first XML element is a LOGINREPLY element with
   // status OK, go to the porfolio screen.  Otherwise,
   // go to the login failure screen and let the user try again.
   if (e.nodeName == "LOGINREPLY" && e.attributes.status == "OK") {
// Save the session ID for future communications with server
   sessionID = e.attributes.session;
// Go to the portfolio viewing screen
      gotoAndStop("portfolioView");
   } else {
      // Login failed! Go to the login failure screen.
      gotoAndStop("loginFailed");
   }
}
```

La première ligne de cette fonction, `var e = this.firstChild`, utilise le mot-clé `this` pour faire référence à l'objet XML `loginReplyXML` qui vient d'être chargé avec XML depuis le serveur. Vous pouvez utiliser `this` car `onLoginReply` a été invoquée sous la forme `loginReplyXML.onLoad`, donc, même si `onLoginReply` se révèle être une fonction ordinaire, elle se comportera quand même comme une méthode de `loginReplyXML`.

Pour envoyer le nom d'utilisateur et le mot de passe sous la forme XML vers le serveur et pour recharger une réponse XML dans l'animation Flash, vous pouvez utiliser la méthode `sendAndLoad` comme dans l'exemple suivant :

```
// C. Send the LOGIN element to the server,
// place the reply in loginReplyXML
   loginXML.sendAndLoad("https://www.imexstocks.com/main.cgi",
loginReplyXML);
```

Pour plus d'informations sur les méthodes XML, reportez-vous à son entrée dans le chapitre 7 « Dictionnaire ActionScript ».

Remarque : Cette démonstration n'est qu'un exemple et nous ne garantissons pas le niveau de sécurité fourni. Si vous appliquez un système sécurisé protégé par mots de passe, assurez-vous de bien comprendre la sécurité de réseau.

Utilisation de l'objet XMLSocket

ActionScript fournit un objet XMLSocket prédéfini qui vous permet d'ouvrir une connexion continue avec un serveur. Une connexion socket permet au serveur de donner l'information au client dès qu'elle est disponible. Sans connexion continue, le serveur devra attendre une requête HTTP. Cette connexion ouverte supprime les périodes d'attente et est souvent utilisée dans des applications en temps réel comme les dialogues en ligne. Les données sont envoyées dans la connexion socket sous forme d'une chaîne et doivent être au format XML. Vous pouvez utiliser l'objet XML pour structurer les données.

Pour créer une connexion socket, vous devez créer une application côté serveur qui attendra la requête de connexion et enverra une réponse à l'animation Flash. Ce type d'applications côté serveur peut être écrit en langage de programmation tel que Java.

Vous pouvez utiliser les méthodes `connect` et `send` de l'objet XMLSocket ActionScript pour transférer le code XML vers et depuis un serveur avec une connexion socket. La méthode `connect` établit une connexion socket avec le port d'un serveur Internet. La méthode `send` transmet un objet XML vers le serveur spécifié dans la connexion socket.

Lorsque vous invoquez la méthode `connect` de l'objet XMLSocket, Flash Player ouvre une connexion TCP/IP vers le serveur et garde cette connexion ouverte jusqu'à ce qu'un des événements suivants arrive :

- La méthode `close` de l'objet XMLSocket est appelée.
- Il n'existe plus aucune référence à l'objet XMLSocket.
- Flash Player est fermé.
- La connexion est coupée (par exemple, le modem est déconnecté).

L'exemple suivant crée une connexion socket XML et envoie les données de l'objet XML `myXML`. Pour comprendre le script, lisez les commentaires de chaque script précédés par les signes `//` :

```
//create a new XMLSocket object
sock = new XMLSocket();
//call its connect method to establish a connection with port 1024
//of the server at the URL
sock.connect("http://www.myserver.com", 1024);
//define a function to assign to the sock object that handles
//the servers response. If the connection succeeds, send the myXML
//object. If it fails, provide an error message in a text field.
function onSockConnect(success){
   if (success){
      sock.send(myXML);
   } else {
      msg="There has been an error connecting to "+serverName;
   }
}
//assign the onSockConnect function to the onConnect property
sock.onConnect = onSockConnect;
```

Pour plus d'informations, reportez-vous à l'entrée sur XMLSocket dans le chapitre 7 « Dictionnaire ActionScript ».

Création de formulaires

Les formulaires de Flash fournissent un type d'interactivité avancé (une combinaison de boutons, d'animations et de champs de texte qui vous permettent de transmettre des informations à une autre application sur un serveur local ou distant). Tous les éléments de formulaires classiques (par exemple, des boutons d'options, des listes déroulantes et des cases à cocher) peuvent être créés sous forme d'animations ou de boutons avec la présentation du style général de votre site Internet. L'élément de formulaire le plus souvent rencontré est un champ d'entrée de texte.

Les types de formulaires communs utilisant de tels éléments d'interface comprennent les interfaces de dialogues en ligne, les bons de commande et les interfaces de recherche. Par exemple, un formulaire Flash peut récupérer des informations concernant les adresses et les envoyer à une autre application qui les compile dans un message électronique ou dans un fichier de base de données. Même un simple champ de texte est considéré comme un formulaire et peut être utilisé pour récupérer les entrées de l'utilisateur et afficher les résultats.

Les formulaires nécessitent deux composants principaux : les éléments d'interface de Flash qui réalisent le formulaire et soit une application côté serveur, soit un script côté client, pour traiter les informations entrées par l'utilisateur. Les étapes suivantes montrent la procédure générale de création d'un formulaire dans Flash.

Pour créer un formulaire :

1. Placez les éléments d'interface dans l'animation en utilisant la mise en page de votre choix.

   Vous pouvez utiliser les éléments d'interface de la bibliothèque commune Boutons - Avancés ou créer les vôtres.

2. Dans le panneau Options du texte, définissez les champs de texte sur Entrée et affectez à chacun un nom unique de variable.

   Pour plus d'informations sur la création de champs texte modifiables, consultez *Utilisation de Flash*.

3. Affectez une action qui, soit envoie, soit charge, soit envoie et charge les données.

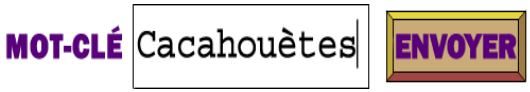
Création d'un formulaire de recherche

Un exemple de formulaire simple est un champ de recherche avec un bouton Envoyer. Comme introduction à la création de formulaires, l'exemple suivant fournit les instructions pour la création d'interface de recherche en utilisant une action `getURL`. En saisissant les informations requises, les utilisateurs peuvent transmettre un mot-clé à un moteur de recherche sur un serveur Internet distant.

Pour créer un simple formulaire de recherche :

1. Créez un bouton pour soumettre les données entrées.
2. Créez une étiquette, un champ de texte vide et une occurrence du bouton sur la scène.

   Votre écran devrait avoir cette apparence :



3. Sélectionnez le champ de texte et choisissez Fenêtre > Panneaux > Options du texte.
4. Dans le panneau Options du texte, définissez les options suivantes :

- Choisissez Texte d'entrée dans le menu contextuel.
- Sélectionnez Bord/Arrière-plan.
- Spécifiez un nom de variable.

   Remarque : Les différents moteurs de recherche peuvent avoir besoin d'un nom de variable spécifique. Allez sur le site Internet du moteur de recherche pour plus de détails.

5. Sur la scène, sélectionnez le bouton et choisissez Fenêtre > Actions.

   Le panneau Actions sur objets apparaît.

   Remarque : Une coche à côté d'Actions dans le menu Fenêtre indique que le panneau est ouvert.

6. Faites glisser l'action `getURL` depuis la boîte à outils sur la fenêtre Script.

7 Dans le panneau Paramètres, définissez les options suivantes :

- Pour URL, entrez l'URL du moteur de recherche.
- Pour Fenêtre, sélectionnez `_blank`. Cela affichera une nouvelle fenêtre qui affichera les résultats de la recherche.
- Pour Variables, sélectionnez Envoyer à l'aide de GET.

8 Pour tester le formulaire, choisissez Fichier > Aperçu avant publication > HTML.

Utilisation de variables dans les formulaires

Vous pouvez utiliser les variables dans un formulaire pour stocker les entrées de l'utilisateur. Pour définir les variables, vous utilisez des champs de texte éditables ou vous affectez des actions à des boutons dans les éléments d'interface. Par exemple, chaque élément d'un menu contextuel est un bouton ayant une action qui définit une variable pour indiquer l'élément sélectionné. Vous pouvez affecter un nom de variable à un champ d'entrée de texte. Le champ de texte se comporte comme une fenêtre qui affiche la valeur de cette variable.

Lorsque vous transmettez des informations à ou depuis un script côté serveur, les variables de l'animation Flash doivent correspondre aux variables du script. Par exemple, si le script attend une variable appelée `password`, le champ de texte dans lequel l'utilisateur entre le mot de passe doit avoir le nom de variable `password`.

Certains scripts nécessitent des variables cachées, variables que l'utilisateur ne voit jamais. Pour créer une variable cachée avec Flash, vous pouvez définir une variable dans une image du clip qui contient les autres éléments du formulaire. Les variables cachées sont envoyées au script côté serveur en même temps que les autres variables définies dans le scénario qui contient l'action qui soumet le formulaire.

Vérification des données entrées

Dans un formulaire qui transmet des variables à une application sur un serveur Internet, vous souhaiterez vérifier que les utilisateurs entrent les informations correctes. Par exemple, vous ne souhaitez pas que les utilisateurs entrent du texte dans un champ de numéro de téléphone. Utilisez une suite d'actions `set variable` en conjonction avec `for` et `if` pour évaluer les données entrées.

L'action, dans l'exemple suivant, vérifie si les données entrées sont des nombres et si les nombres sont écrits selon le format ###-###-####. Si les données sont correctes, le message suivant s'affiche : « Good, this is a valid phone number! » . Si les données sont incorrectes, le message « This phone number is invalid! » s'affiche.

Pour utiliser ce script dans une animation, créez deux champs de texte sur la scène et choisissez Entrée dans le panneau Options du texte pour chacun. Affectez la variable `phoneNumber` à un champ de texte et affectez la variable `message` à l'autre. Attachez l'action suivante à un bouton sur la scène près des champs de texte :

```
on (release) {
   valid = validPhoneNumber(phoneNumber);
   if (valid) {
      message = "Good, this is a valid phone number!";
   } else {
      message = "This phone number is invalid!";
   }
   function isdigit(ch) {
      return ch.length == 1 && ch >= '0' && ch <= '9';
   }
   function validPhoneNumber(phoneNumber) {
      if (phoneNumber.length != 12) {
         return false;
      }
      for (var index = 0; index < 12; index++) {
         var ch = phoneNumber.charAt(index);
         if (index == 3 || index == 7) {
            if (ch != "-") {
            return false;
            }
         } else if (!isdigit(ch)) {
            return false;
         }
      }
      return true;
   }
}
```

Pour envoyer les données, créez un bouton ayant une action similaire à l'exemple suivant (remplacez les arguments `getURL` par les arguments correspondants à votre animation).

```
on (release) {
   if (valid) {
      getURL("http://www.webserver.com", "_self", "GET");
   }
}
```

Pour plus d'informations sur ces instructions ActionScript, voir `set`, `for` et `if` dans le chapitre 7, « Dictionnaire ActionScript » à la page 173.

Envoi de messages vers et depuis Flash Player

Pour envoyer des messages depuis une animation Flash vers son environnement hôte (par exemple, un navigateur Internet, une animation Director ou Flash Player autonome), vous pouvez utiliser l'action `fscommand`. Cette action vous permet d'améliorer votre animation en utilisant les capacités de l'hôte. Par exemple, vous pouvez transmettre une action `fscommand` à une fonction JavaScript dans une page HTML qui ouvre une nouvelle fenêtre de navigateur ayant des propriétés spécifiques.

Pour contrôler une animation dans Flash Player depuis les langages de script du navigateur Internet (par exemple, JavaScript, VBScript et Microsoft Jscript), vous pouvez utiliser les méthodes Flash Player (fonctions qui envoient des messages depuis un environnement hôte vers l'animation Flash). Par exemple, vous pouvez avoir un lien avec une page HTML qui envoie votre animation Flash vers une image spécifique.

Utilisation de l'action fscommand

Utilisez l'action `fscommand` pour envoyer un message au programme, quel qu'il soit, hébergeant Flash Player. L'action `fscommand` possède deux paramètres : *`command`* et *`arguments`*. Pour envoyer un message à la version autonome du Flash Player, vous devez utiliser des commandes et des arguments prédéfinis. Par exemple, l'action suivante définit le lecteur autonome pour qu'il affiche l'animation en taille plein écran lorsque le bouton est relâché :

```
on(release){
   fscommand("fullscreen", "true");
}
```

Le tableau suivant indique les valeurs que vous pouvez spécifier aux paramètres *commande* et *arguments* de l'action fscommand pour contrôler une animation se déroulant dans le lecteur autonome (y compris les projecteurs) :

commande	*arguments*	**Effet**
quit	Aucun	Ferme le projection.
fullscreen	true ou false	Spécifier true définit Flash Player en mode plein écran. Spécifier false renvoie le lecteur en affichage normal du menu.
allowscale	true ou false	Spécifier false définit le lecteur de sorte qu'il soit toujours affiché dans sa taille originale et que son échelle ne soit jamais modifiée. Spécifier true oblige l'animation à adopter l'échelle 100 % du lecteur.
showmenu	true ou false	Spécifier true affiche le jeu complet des éléments du menu contextuel. Spécifier false masque tous les éléments du menu contextuel, sauf À propos de Flash Player.
exec	Chemin vers l'application	Exécute une application depuis le projecteur.

Pour utiliser fscommand pour envoyer un message à un langage de script (par exemple, JavaScript dans un navigateur Internet), vous pouvez transmettre deux arguments quelconques dans les paramètres *command* et *arguments*. Ces arguments peuvent être des chaînes ou des expressions et seront utilisés dans une fonction JavaScript qui traite, l'action fscommand.

Une action fscommand invoque la fonction JavaScript *nomAnimation*_DoFSCommand dans la page HTML qui intègre l'animation Flash, où *nomAnimation* est le nom de Flash Player tel qu'il est affecté par l'attribut NAME de la balise EMBED ou par l'attribut ID de la balise OBJECT. Si le nom myMovie a été affecté au Flash Player, la fonction JavaScript invoquée est myMovie_DoFSCommand.

Pour utiliser l'action fscommand pour ouvrir une boîte de messages depuis une animation Flash dans la page HTML avec JavaScript :

1 Dans la page HTML qui intègre l'animation Flash, ajoutez le code JavaScript suivant :

```
function theMovie_DoFSCommand(command, args) {
    if (command == "messagebox") {
       alert(args);
    }
}
```

Si vous publiez votre animation en utilisant Flash avec le modèle FSCommand dans les Paramètres de publication HTML, ce code est inséré automatiquement. Les attributs `NAME` et `ID` de l'animation seront les noms de fichiers. Par exemple, pour le fichier `myMovie.fla`, les attributs seront définis par `myMovie`. Pour plus d'informations sur la publication, voir *Utilisation de Flash.*

2 Dans l'animation Flash, ajoutez l'action `fscommand` à un bouton :

```
fscommand("messagebox", "This is a message box invoked from
within Flash.")
```

Vous pouvez également utiliser des expressions pour l'action et les arguments `fscommand`, comme dans l'exemple suivant :

```
fscommand("messagebox", "Hello, " & name & ", welcome to our
Web site!")
```

3 Choisissez Fichier > Aperçu avant publication > HTML pour tester l'animation.

L'action `fscommand` peut envoyer des messages à Macromedia Director qui sont interprétés par Lingo comme des chaînes, des événements ou un code exécutable Lingo. Si le message est une chaîne ou un événement, vous devez écrire le code Lingo pour le recevoir depuis l'action `fscommand` et entraîner une action dans Director. centre d'assistance de Director à l'adresse http://www.macromedia.com/support/director.

En Visual Basic, Visual C++, et dans d'autres programmes pouvant héberger les contrôles ActiveX, `fscommand` envoie un événement VB avec deux chaînes qui peut être traité dans l'environnement du langage de programmation. Pour plus d'informations, utilisez le mot-clé `Flash method` pour effectuer une recherche sur le centre d'assistance de Flash à l'adresse http://www.macromedia.com/support/flash.

À propos des méthodes Flash Player

Vous pouvez utiliser les méthodes Flash Player pour contrôler les animations dans Flash Player depuis les langages de script des navigateurs Internet (par exemple, JavaScript et VBScript). Comme pour les autres méthodes, vous pouvez utiliser les méthodes Flash Player pour envoyer des appels à des animations Flash Player depuis un environnement de script autre que ActionScript. Chaque méthode possède un nom et la plupart prennent des arguments. Un argument spécifie une valeur sur laquelle opère la méthode. Le calcul effectué par certaines méthodes renvoie une valeur qui peut être utilisée par un environnement de script.

Deux technologies différentes permettent de communiquer entre le navigateur et Flash Player : LiveConnect (Netscape Navigator 3.0 et versions suivantes sous Windows 95/98/2000/NT ou Power Macintosh) et ActiveX (Microsoft Internet Explorer 3.0 et versions suivantes sous Windows 95/98/2000/NT). Bien que les techniques de script soient équivalentes pour tous les navigateurs et les langages, des propriétés et des événements supplémentaires sont disponibles pour l'utilisation des contrôles ActiveX.

Pour plus d'informations, y compris une liste complète des méthodes de script Flash Player, utilisez le mot-clé `Flash method` pour effectuer une recherche sur le centre d'assistance Flash à l'adresse http://www.macromedia.com/support/flash.

CHAPITRE 6
Dépannage d'ActionScript

Le degré de perfectionnement de certaines actions, surtout lorsqu'elles sont combinées les unes aux autres, peut entraîner une complexité des animations Flash. Comme avec tout langage de programmation, vous pouvez écrire un script ActionScript incorrect qui provoque des erreurs dans vos scripts. Utiliser de bonnes techniques de programmation facilite le dépannage de l'animation lorsqu'apparaît un comportement inattendu.

Flash dispose de plusieurs outils vous permettant de tester vos animations en mode test d'animation ou sur un navigateur web. Le Débogueur montre une liste d'affichage hiérarchique des clips actuellement chargés dans Flash Player. Il vous permet également d'afficher ou de modifier des valeurs de variables pendant la lecture de l'animation. En mode test d'animation, la fenêtre Résultat affiche des messages d'erreur et la liste des variables et des objets. Vous pouvez également utiliser l'action `trace` dans vos scripts pour envoyer les notes de programmation et les valeurs des expressions dans la fenêtre Résultat.

Instructions de programmation et de dépannage

Si vous utilisez de bonnes techniques de programmation lorsque vous écrivez des scripts, vos animations contiendront moins de bogues (erreurs de programmation). Vous pouvez utiliser les instructions suivantes pour éviter les problèmes et pour les corriger s'ils se produisent.

Utilisation de bonnes techniques de programmation

Il est conseillé d'enregistrer plusieurs versions de l'animation au cours de votre travail. Sélectionnez Fichier > Enregistrer sous pour enregistrer une version sous un nom différent toutes les demi-heures. Vous pouvez utiliser l'historique des versions pour déterminer quand un problème surgit en recherchant le fichier le plus récent ne contenant pas le problème. Avec cette approche, vous disposerez toujours d'une version fonctionnelle, même si un fichier est altéré.

Une autre technique de programmation consiste à effectuer des tests de manière anticipée, souvent et sur toutes les plates-formes cibles afin de détecter les problèmes dès qu'ils surgissent. Sélectionnez Contrôle > Tester l'animation pour exécuter votre animation en mode test d'animation lorsque vous effectuez une modification importante ou avant d'enregistrer une version. En mode test d'animation, l'animation s'exécute dans une version du lecteur autonome.

Si le public cible doit visualiser l'animation sur le web, il est important de tester également cette dernière sur un navigateur. Dans certaines situations (par exemple, si vous développez un site Intranet), vous pouvez connaître le navigateur et la plate-forme utilisés par le public cible. Si vous développez un site web, testez votre animation dans tous les navigateurs sur toutes les plates-formes possibles.

Il est conseillé d'utiliser ces techniques de programmation :

- Utilisez l'action `trace` pour envoyer des commentaires dans la fenêtre Résultat. Voir « Utilisation de trace » à la page 171.
- Utilisez l'action `comment` pour inclure des notes instructives qui apparaissent uniquement dans le panneau Actions. (Voir « Commentaires » à la page 53.)
- Utilisez une convention d'affectation des noms cohérente pour identifier les éléments d'un script. Par exemple, il est conseillé d'éviter les espaces dans les noms. Les noms de fonctions et de variables doivent commencer par une minuscule et chaque nouveau mot par une majuscule (`myVariableName`, `myFunctionName`). Les fonctions de construction commencent par une majuscule (`MyConstructorFunction`). Le plus important est de choisir un style explicite et de le conserver par la suite.
- Utilisez des noms de variables explicites qui reflètent le type des informations qu'elles contiennent. Par exemple, une variable contenant des informations sur le dernier bouton utilisé peut être nommée `lastButtonPressed`. Le nom `foo` permet difficilement de se souvenir du contenu de la variable.

- Utilisez des champs de texte modifiables dans des calques-guides pour assurer le suivi des valeurs des variables au lieu d'utiliser le Débogueur.
- Utilisez l'Explorateur d'animation en mode modification d'animation pour visualiser la liste d'affichage et toutes les actions d'une animation. Voir Aide de Flash.
- Utilisez l'action `for...in` pour effectuer une boucle sur les propriétés des clips, dont les clips enfants. Vous pouvez utiliser l'action `for...in` avec l'action `trace` pour envoyer une liste de propriétés dans la fenêtre Résultat. Voir « Répétition d'une action » à la page 72.

Utilisation d'une liste de contrôle de dépannage

Comme dans tout environnement de scripts, les créateurs de scripts réalisent souvent certaines erreurs. La liste suivante est idéale pour commencer le dépannage de l'animation :

- Assurez-vous d'être en mode test d'animation.

  Seuls les boutons et actions d'image simples (par exemple, `gotoAndPlay` et `stop`) fonctionneront en mode programmation. Sélectionnez Contrôle > Activer les actions d'image simples ou Contrôle > Activer les boutons simples pour activer ces actions.
- Assurez-vous qu'il n'existe aucune action d'image sur plusieurs calques en conflit.
- Si vous travaillez avec le panneau Actions en mode Normal, assurez-vous que l'instruction prend la valeur Expression.

  Si vous transmettez une expression dans une action et que vous n'avez pas sélectionné la case Expression, la valeur sera transmise en tant que chaîne. Voir « Utilisation des opérateurs pour manipuler les valeurs des expressions » à la page 62.
- Assurez-vous que des éléments ActionScript ne portent pas le même nom.

  Il est conseillé de donner un nom unique à chaque variable, fonction, objet et propriété. Les variables locales sont des exceptions, elles doivent seulement être uniques dans leur portée et sont souvent réutilisées en tant que compteurs. Voir « Portée d'une variable » à la page 59.

Pour obtenir plus de conseils sur le dépannage d'une animation Flash, reportez-vous à Flash Support Center à l'adresse http://www.macromedia.com/support/flash.

Utilisation du Débogueur

Le Débogueur vous permet de rechercher des erreurs dans une animation en cours d'exécution dans Flash Player. Vous pouvez visualiser la liste d'affichage des clips et des animations chargées et modifier les valeurs des variables et propriétés afin de déterminer les valeurs correctes. Vous pouvez alors revenir aux scripts et les modifier afin qu'ils produisent les résultats corrects. Pour utiliser le Débogueur, vous devez exécuter Flash Debug Player, une version spéciale de Flash Player.

Flash Debug Player installe automatiquement l'application de programmation Flash 5. Il vous permet de télécharger la liste d'affichage, les paires nom/valeur de variable et les paires nom/valeur de propriété dans le Débogueur dans l'application de programmation Flash.

Pour afficher le Débogueur :

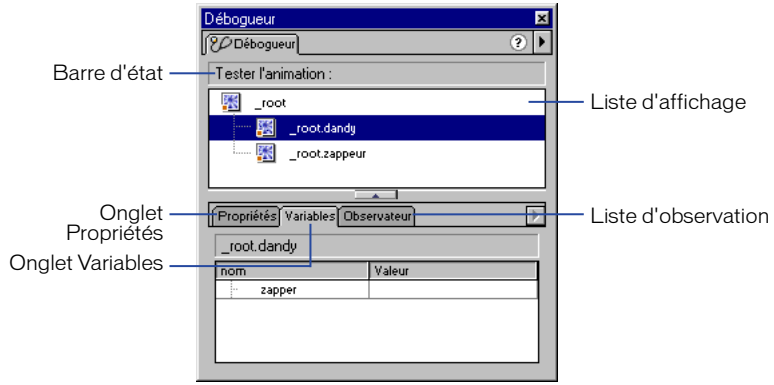
Sélectionnez Fenêtre > Débogueur.

Le Débogueur est ouvert à l'état inactif. Aucune information n'apparaît dans la liste d'affichage tant qu'une commande n'est pas exécutée à partir de Flash Player.

Pour activer le Débogueur en mode test d'animation :

Sélectionnez Contrôle > Déboguer l'animation.

Le Débogueur est ouvert à l'état actif.



Activation du débogage d'une animation

Lors de l'exportation d'une animation Flash Player, vous pouvez choisir d'activer le débogage de l'animation et de créer un mot de passe de débogage. Si vous n'activez pas le débogage, le Débogueur ne s'active pas.

Comme dans JavaScript ou HTML, toute variable ActionScript côté client peut potentiellement être visualisée par l'utilisateur. Pour stocker les variables de façon sûre, vous devez les envoyer à une application côté serveur au lieu de les stocker dans une animation.

Cependant, en tant que développeur Flash, vous disposez peut-être d'autres secrets professionnels, tels que les structures de clips, que vous ne voulez pas révéler. Pour vous assurer que seuls des utilisateurs reconnus puissent regarder vos animations avec Flash Debug Player, vous pouvez publier l'animation avec un mot de passe Débogueur.

Pour activer le débogage et créer un mot de passe :

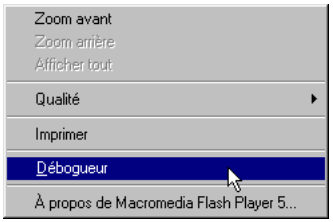
1. Choisissez Fichier > Paramètres de publication.
2. Cliquez sur l'onglet Flash.
3. Sélectionnez Débogage autorisé.
4. Pour définir un mot de passe, entrez-le dans la zone Mot de passe.

   Sans mot de passe, vous ne pouvez pas télécharger d'informations vers le Débogueur. Si vous ne renseignez pas le champ de mot de passe, aucun mot de passe n'est requis.

Pour activer le Débogueur dans un navigateur web :

1. Cliquez avec le bouton droit de la souris (Windows) ou cliquez tout en appuyant sur la touche Ctrl (Macintosh) pour ouvrir le menu contextuel de Flash Debug Player.
2. Sélectionnez Débogueur.

Remarque : Vous pouvez utiliser le Débogueur pour contrôler une seule animation à la fois. Flash doit être ouvert pour pouvoir utiliser le Débogueur.



Menu contextuel de Flash Debug Player

À propos de la barre d'état

Une fois activée, la barre d'état du Débogueur affiche l'URL ou le chemin d'accès local de l'animation. Flash Player est mis en œuvre sous différentes formes en fonction de l'environnement de lecture. La barre d'état du Débogueur affiche le type de Flash Player exécutant l'animation :

- Mode test d'animation
- Lecteur autonome
- Plug-in Netscape

  Le plug-in Netscape est utilisé avec Netscape Navigator sous Windows et Macintosh et avec Microsoft Internet Explorer sous Macintosh.
- Contrôle ActiveX

  Le contrôle ActiveX est utilisé avec Internet Explorer sous Windows.

À propos de la liste d'affichage

Lorsqu'il est actif, le Débogueur montre une vue réelle de la liste d'affichage de clips. Vous pouvez développer ou réduire des branches pour visualiser tous les clips actuellement sur la scène. Lorsque des clips sont ajoutés à l'animation, ou supprimés, la liste d'affichage est immédiatement mise à jour. Vous pouvez redimensionner la liste d'affichage en déplaçant le séparateur horizontal ou en déplaçant le coin inférieur droit.

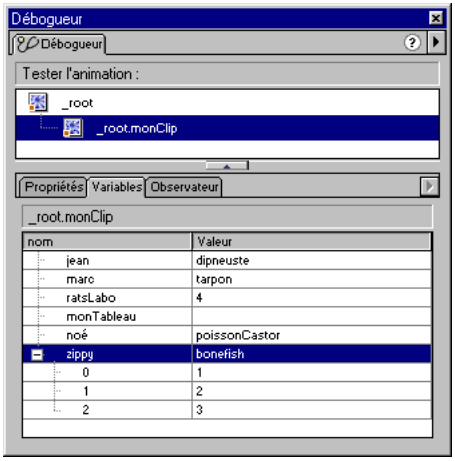
Affichage et modification de variables

L'onglet Variables du Débogueur affiche les noms et valeurs de toutes les variables de l'animation. Si vous modifiez la valeur d'une variable dans l'onglet Variables, vous pouvez voir la modification dans l'animation en cours d'exécution. Par exemple, pour tester la détection de conflit dans un jeu, vous pouvez entrer la valeur de la variable afin de positionner une balle à l'emplacement correct près d'un mur.

Pour afficher une variable :

1. Sélectionnez le clip contenant la variable provenant de la liste d'affichage.
2. Cliquez sur l'onglet Variables.

La liste d'affichage est automatiquement mise à jour au cours de la lecture de l'animation. Un clip supprimé de l'animation au niveau d'une image spécifique est également supprimé de la liste d'affichage du Débogueur ; cela entraîne la suppression de la valeur et du nom de la variable.



Pour modifier la valeur d'une variable :

Sélectionnez la valeur et entrez-en une nouvelle.

La valeur doit être une valeur constante (par exemple, `"Hello"`, `3523`, ou `"http://www.macromedia.com"`) et non une expression (par exemple, `x + 2`, ou `eval("name:" +i)`). La valeur peut être une chaîne (n'importe quelle valeur comprise entre guillemets (`""`), un nombre ou une valeur booléenne (`true` ou `false`).

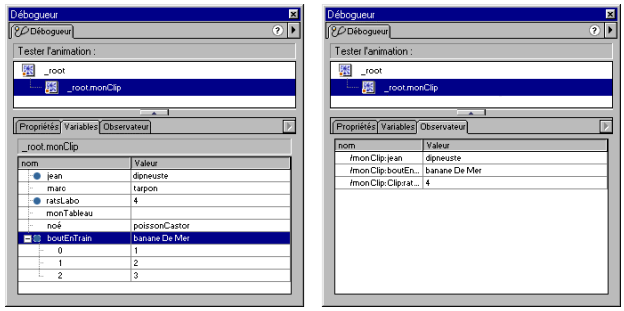
Les variables Object et Array sont affichées sous l'onglet Variables. Cliquez sur le bouton Ajouter (+) pour visualiser leurs propriétés et valeurs. Cependant, vous ne pouvez pas entrer de valeurs pour Object ou Array (par exemple, `{name: "I am an object"}` or `[1, 2, 3]`) dans les champs correspondants.

Remarque : Pour extraire la valeur d'une expression en mode test d'animation, utilisez l'action `trace`. Voir « Utilisation de trace » à la page 171.

Utilisation de la liste d'observation

Pour contrôler facilement un ensemble de variables critiques, vous pouvez marquer les variables qui doivent apparaître dans la liste d'observation. Cette liste affiche le chemin d'accès absolu de la variable et la valeur. Vous pouvez également entrer une nouvelle valeur de variable dans la liste d'observation.

Seules des variables peuvent être ajoutées à la liste d'observation et non des propriétés ou des fonctions.



Variables marquées pour la liste d'observation et variables dans la liste d'observation.

Pour ajouter des variables à la liste d'observation, procédez de l'une des manières suivantes :

- Sous l'onglet Variables, cliquez avec le bouton droit de la souris (Windows) ou cliquez tout en appuyant sur la touche Ctrl (Macintosh) sur une variable sélectionnée, puis sélectionnez Observateur dans le menu contextuel. Un point bleu apparaît en regard de la variable.
- Sous l'onglet Observateur, cliquez avec le bouton droit de la souris (Windows) ou cliquez tout en appuyant sur la touche Ctrl (Macintosh) sur une variable sélectionnée, puis sélectionnez Ajouter dans le menu contextuel. Entrez le nom et la valeur de la variable dans les champs.

Pour supprimer des variables de la liste d'observation :

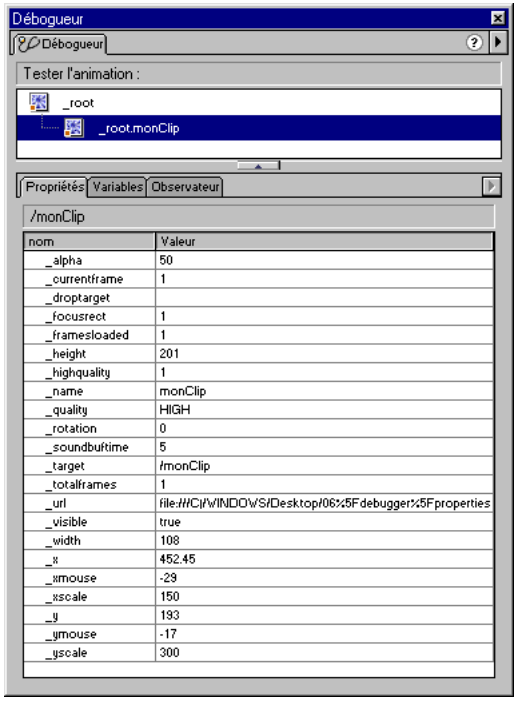
Sous l'onglet Observateur, cliquez avec le bouton droit de la souris (Windows) ou cliquez tout en appuyant sur la touche Ctrl (Macintosh) sur une variable sélectionnée, puis sélectionnez Remove dans le menu contextuel.

Affichage de propriétés d'animation et modification de propriétés éditables

L'onglet Propriétés du Débogueur affiche toutes les valeurs de propriétés de tout clip sur la scène. Vous pouvez modifier la valeur d'une propriété et voir la modification dans l'animation en cours d'exécution. Certaines propriétés de clip sont en lecture seule et ne peuvent pas être modifiées.

Pour afficher les propriétés d'un clip :

1. Sélectionnez un clip dans la liste des affichages.
2. Cliquez sur l'onglet Propriétés.



Pour modifier la valeur d'une propriété :

Sélectionnez la valeur et entrez-en une nouvelle.

La valeur doit être une valeur constante (par exemple, `50`, ou `"clearwater"`) plutôt qu'une expression (par exemple, `x + 50`). La valeur peut être une chaîne (n'importe quelle valeur comprise entre guillemets (`""`), un nombre ou une valeur booléenne (`true` ou `false`). Vous ne pouvez pas entrer de valeurs pour Object ou Array (par exemple, `{id: "rogue"}` ou `[1, 2, 3]`) dans les champs correspondants.

Pour plus d'informations, voir « Chaîne » à la page 54 et « Utilisation des opérateurs pour manipuler les valeurs des expressions » à la page 62.

Remarque : Pour extraire la valeur d'une expression en mode test d'animation, utilisez l'action `trace`. Voir « Utilisation de trace » à la page 171.

Utilisation de la fenêtre Résultat

En mode test d'animation, la fenêtre Résultat affiche des informations facilitant le dépannage de votre animation. Certaines informations, telles que les erreurs de syntaxe, sont automatiquement affichées. Vous pouvez afficher d'autres informations à l'aide des commandes Lister les objets et Lister les variables. Voir « Utilisation de Lister les objets » à la page 169 et « Utilisation de Lister les variables » à la page 170.

Si vous utilisez l'action `trace` dans vos scripts, vous pouvez envoyer des informations spécifiques dans la fenêtre Résultat au cours de l'exécution de l'animation. Il peut s'agir de notes relatives à l'état de l'animation ou à la valeur d'une expression. Voir « Utilisation de trace » à la page 171.

Pour afficher la fenêtre Résultat :

1. Si l'animation n'est pas en cours d'exécution en mode test d'animation, sélectionnez Contrôle > Tester l'animation.
2. Sélectionnez Fenêtre > Résultat.

   La fenêtre Résultat apparaît.

Remarque : La fenêtre Résultat apparaît automatiquement si un script contient des erreurs de syntaxe.

3 Pour utiliser le contenu de la fenêtre Résultat, utilisez le menu Options :

- Sélectionnez Options > Copier pour copier le contenu de la fenêtre Résultat dans le Presse-papiers.
- Sélectionnez Options > Effacer pour effacer le contenu de la fenêtre.
- Sélectionnez Options > Enregistrer dans un fichier pour enregistrer le contenu de la fenêtre dans un fichier texte.
- Sélectionnez Options > Imprimer pour imprimer le contenu de la fenêtre.

Utilisation de Lister les objets

En mode test d'animation, la commande Lister les objets affiche le niveau, l'image, le type d'objet (forme, clip ou bouton) et le chemin cible d'une occurrence de clip dans une liste hiérarchique. Cela est particulièrement utile pour rechercher le chemin cible correct et le nom de l'occurrence. Contrairement au Débogueur, la liste ne se met pas automatiquement à jour au cours de la lecture de l'animation ; vous devez sélectionner la commande Lister les objets chaque fois que vous voulez envoyer les informations à la fenêtre Résultat.

Pour afficher une liste d'objets dans une animation :

1 Si l'animation n'est pas en cours d'exécution en mode test d'animation, sélectionnez Contrôle > Tester l'animation.

2 Sélectionnez Déboguer > Lister les objets.

La liste des objets actuellement en scène s'affiche dans la fenêtre Résultat, comme dans l'exemple suivant :

```
Layer #0: Frame=3
Movie Clip: Frame=1 Target=_root.MC
      Shape:
   Movie Clip: Frame=1 Target=_root.instance3
      Shape:
   Button:
      Movie Clip: Frame=1 Target=_root.instance3.instance2
         Shape:
```

Remarque : Dans ce contexte, un objet est pris en compte comme une forme ou un symbole sur la scène.

Utilisation de Lister les variables

En mode test d'animation, la commande Lister les variables affiche une liste de toutes les variables actuelles de l'animation. Cela est particulièrement utile pour rechercher la variable cible correcte et le nom de la variable. Contrairement au Débogueur, la liste ne se met pas automatiquement à jour au cours de la lecture de l'animation ; vous devez sélectionner la commande Lister les variables chaque fois que vous voulez envoyer les informations à la fenêtre Résultat.

Pour afficher une liste de variables dans une animation :

1. Si l'animation n'est pas en cours d'exécution en mode test d'animation, sélectionnez Contrôle > Tester l'animation.
2. Sélectionnez Déboguer > Lister les variables.

La liste des variables actuelles de l'animation s'affiche dans la fenêtre Résultat, comme dans l'exemple suivant :

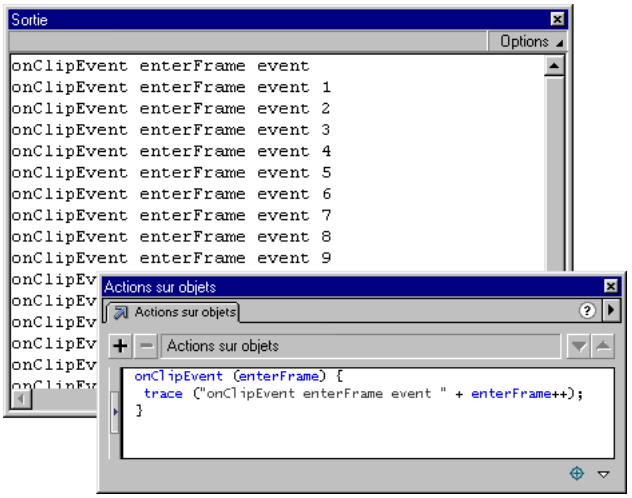
```
Level #0:
   Variable _root.country = "Sweden"
   Variable _root.city = "San Francisco"
Movie Clip: Target=""
Variable _root.instance1.firstName = “Rick”
```

Utilisation de trace

Si vous utilisez l'action `trace` dans un script, vous pouvez envoyer des informations dans la fenêtre Résultat. Par exemple, tout en testant une animation ou une scène, vous pouvez envoyer des notes de programmation spécifiques à la fenêtre ou afficher des résultats spécifiques en appuyant sur un bouton ou en lisant une image. L'action `trace` est identique à l'instruction JavaScript `alert`.

Si vous utilisez l'action `trace` dans un script, vous pouvez utiliser des expressions en tant qu'instructions. La valeur d'une expression est affichée dans la fenêtre Résultat en mode test d'animation, comme dans l'exemple suivant :

```
onClipEvent(enterFrame){
   trace("onClipEvent enterFrame" + enterFrame++)
}
```



L'action `trace` renvoie les valeurs qui sont affichées dans la fenêtre Résultat.

CHAPITRE 7
Dictionnaire ActionScript

Cette partie du *Guide de référence ActionScript* décrit la syntaxe et l'utilisation des éléments ActionScript dans Flash 5 et les versions suivantes. Les entrées de ce guide sont les mêmes que celles de l'Aide du dictionnaire ActionScript. Pour utiliser les exemples dans un script, copiez le texte de l'exemple dans l'Aide du dictionnaire ActionScript et collez-le dans le panneau Actions en mode Expert.

Ce dictionnaire répertorie l'ensemble des éléments ActionScript (opérateurs, mots-clés, instructions, actions, propriétés, fonctions, objets et méthodes). Pour une vue d'ensemble de toutes les entrées du dictionnaire, voir « Contenu du dictionnaire » à la page 176 ; les tableaux de cette section sont un bon point de départ pour rechercher des objets ou des méthodes symboliques dont vous ne connaissez pas la classe d'objet.

ActionScript respecte la norme ECMA-262 (spécification écrite par la European Computer Manufacturers Association), sauf remarques contraires.

Ce dictionnaire comporte deux types d'entrées différents :

- Des entrées individuelles pour les opérateurs, les mots-clés, les fonctions, les variables, les propriétés, les méthodes et les instructions
- Des entrées d'objets qui fournissent des détails généraux sur les objets prédéfinis

Utilisez les informations contenues dans l'aperçu des entrées pour interpréter la structure et les conventions utilisées dans ces deux types d'entrées.

Aperçu d'une entrée pour la plupart des éléments ActionScript

L'aperçu suivant des entrées du dictionnaire explique les conventions utilisées pour tous les éléments ActionScript qui ne sont pas des objets.

Titre de l'entrée

Toutes les entrées sont classées dans l'ordre alphabétique. Cet ordre ne tient pas compte des majuscules, des caractères d'interlignes soulignés, et ainsi de suite.

Syntaxe

La section « Syntaxe » fournit la syntaxe correcte pour l'utilisation des éléments ActionScript dans votre code. La partie de code de la syntaxe est en `code font (police du code)` et les arguments que vous devez fournir sont en *`police du code italique`*. Les parenthèses indiquent les arguments optionnels.

Arguments

Cette section décrit les arguments répertoriés dans la syntaxe.

Description

Cette section identifie l'élément (par exemple, comme opérateur, méthode, fonction ou autre) et décrit son utilisation.

Lecteur

Cette section indique les versions de lecteur qui supportent l'élément. Ceci est différent de la version de Flash utilisée pour développer un contenu. Par exemple, si vous créez un contenu pour Flash 4 Player en utilisant l'outil de programmation de Flash 5, vous ne pouvez pas utiliser les éléments ActionScript qui sont seulement disponibles pour Flash 5 Player.

Avec l'introduction de ActionScript Flash 5, certains éléments ActionScript de Flash 4 (et versions antérieures) ont été désapprouvés. Bien que ces éléments soient toujours supportés par Flash 5 Player, il est recommandé d'utiliser les nouveaux éléments Flash 5.

De plus, la fonctionnalité des opérateurs a été nettement améliorée dans Flash 5. Non seulement de nombreux opérateurs mathématiques ont été ajoutés, mais certains des anciens opérateurs sont maintenant capables de manipuler des types de données supplémentaires. Pour maintenir la cohésion des types de données, les fichiers Flash 4 sont automatiquement modifiés lorsqu'ils sont importés dans l'environnement de programmation de Flash 5, mais ces modifications n'affectent pas la fonctionnalité du script original. Pour plus d'informations, voir les entrées de + (addition), < (inférieur à), > (supérieur à), <= (inférieur ou égal à), >= (supérieur ou égal à), != (inégalité) et = (égalité).

Exemple

Cette section fournit un échantillon de code montrant l'utilisation de l'élément.

Voir aussi

Cette section répertorie les entrées du dictionnaire ActionScript connexes.

Aperçu d'une entrée pour les objets

L'aperçu suivant des entrées du dictionnaire explique les conventions utilisées pour les objets prédéfinis ActionScript. Les objets sont classés par ordre alphabétique avec tous les autres éléments du dictionnaire.

Titre de l'entrée

Le titre de l'entrée fournit le nom de l'objet. Le nom de l'objet est suivi d'un paragraphe contenant les informations générales relatives à l'objet.

Tableaux résumant les méthodes et les propriétés

Chaque entrée d'objet contient un tableau répertoriant toutes les méthodes associées à l'objet. Si l'objet possède des propriétés (souvent des constantes), ces éléments sont répertoriés dans un tableau supplémentaire. Toutes les méthodes et propriétés indiquées dans ces tableaux possèdent leurs propres entrées dans le dictionnaire, à la suite de l'entrée de l'objet.

Constructeur

Si l'objet nécessite que vous utilisiez un constructeur pour accéder à ses méthodes et propriétés, le constructeur est décrit à la fin de l'entrée de l'objet. Cette description possède tous les éléments standard (description de la syntaxe et ainsi de suite) des autres entrées du dictionnaire.

Listes des méthodes et propriétés

Les méthodes et les propriétés d'un objet sont classées par ordre alphabétique après l'entrée de l'objet.

Contenu du dictionnaire

Toutes les entrées du dictionnaire sont classées par ordre alphabétique. Cependant, certains opérateurs étant des symboles, ils sont classés dans l'ordre ASCII. De plus, les méthodes associées à un objet sont répertoriées avec le nom de l'objet (par exemple, la méthode `abs` de l'objet Math est répertoriée sous la forme `Math.abs`).

Les deux tableaux suivants vous aideront à localiser ces éléments. Le premier tableau répertorie les opérateurs symboliques selon leur ordre d'apparition dans le dictionnaire. Le second tableau répertorie tous les autres éléments ActionScript.

Remarque : Pour l'ordre de priorité et l'associativité des opérateurs, voir l'Annexe A.

Opérateurs symboliques	
--	(décrémentation)
++	(incrémentation)
!	(NOT logique)
!=	(inégalité)
%	(pourcentage)
%=	(affectation de pourcentage)
&	(AND au niveau du bit)
&&	(AND court-circuit)
&=	(affectation AND au niveau du bit)
( )	(parenthèses)
-	(moins)
*	(multiplication)
*=	(affectation de multiplication)
,	(virgule)
.	(point)
? :	(conditionnel)
/	(division)
//	(délimiteur de commentaires)
/*	(délimiteur de commentaires)
/=	(affectation de division)

Opérateurs symboliques	
[]	(accès tableau)
^	(XOR au niveau du bit)
^=	(affectation XOR au niveau du bit)
{}	(initialisateur d'objet)
|	(OR au niveau du bit)
||	(OR logique)
|=	(affectation OR au niveau du bit)
~	(NOT au niveau du bit)
+	(addition)
+=	(affectation d'addition)
<	(inférieur à)
<<	(décalage gauche au niveau du bit)
<<=	(décalage gauche au niveau du bit et affectation)
<=	(inférieur ou égal à)
<>	(inégalité)
=	(affectation)
-=	(affectation de négation)
==	(égalité)
>	(supérieur à)
>=	(supérieur ou égal à)
>>	(décalage droit au niveau du bit)
>>=	(décalage droit au niveau du bit et affectation)
>>>	(décalage droit non signé au niveau du bit)
>>>=	(décalage droit non signé au niveau du bit et affectation)

Le tableau suivant répertorie tous les éléments ActionScript qui ne sont pas des opérateurs symboliques.

Élément ActionScript	Voir entrée
abs	« Math.abs » à la page 298
acos	« Math.acos » à la page 299
add	« add » à la page 222
and	« and » à la page 223
_alpha	« _alpha » à la page 222
appendChild	« XML.appendChild » à la page 396
Array	« Tableau (objet) » à la page 223
asin	« Math.asin » à la page 299
atan	« Math.atan » à la page 300
atan2	« Math.atan2 » à la page 300
attachMovie	« MovieClip.attachMovie » à la page 314
attachSound	« Sound.attachSound » à la page 360
attributes	« XML.attributes » à la page 396
BACKSPACE	« Key.BACKSPACE » à la page 284
Booléen	« Boolean (fonction) » à la page 233, « Booléen (objet) » à la page 234
break	« break » à la page 235
call	« call » à la page 236
CAPSLOCK	« Key.CAPSLOCK » à la page 284
ceil	« Math.ceil » à la page 300
charAt	« String.charAt » à la page 373
charCodeAt	« String.charCodeAt » à la page 374
childNodes	« XML.childNodes » à la page 397
chr	« chr » à la page 236
cloneNode	« XML.cloneNode » à la page 397
close	« XMLSocket.close » à la page 413
Couleur	« Couleur (objet) » à la page 237

Élément ActionScript	Voir entrée
concat	« Array.concat » à la page 226, « String.concat » à la page 374
connect	« XMLSocket.connect » à la page 413
constructor	Chaîne, Booléen, Couleur, Date, Nombre, Objet, Son, Chaîne, XML, XMLSocket
continue	« continue » à la page 241
CONTROL	« Key.CONTROL » à la page 284
cos	« Math.cos » à la page 301
createElement	« XML.createElement » à la page 398
createTextNode	« XML.createTextNode » à la page 398
_currentframe	« _currentframe » à la page 242
Date	« Date (objet) » à la page 242
delete	« delete » à la page 260
DELETEKEY	« Key.DELETEKEY » à la page 285
docTypeDecl	« XML.docTypeDecl » à la page 399
do...while	« do...while » à la page 261
DOWN	« Key.DOWN » à la page 285
_droptarget	« _droptarget » à la page 262
duplicateMovieClip	« duplicateMovieClip » à la page 263, « MovieClip.duplicateMovieClip » à la page 314
E	« Math.E » à la page 301
else	« else » à la page 264
END	« Key.END » à la page 285
ENTER	« Key.ENTER » à la page 286
eq	« eq (égal–propre aux chaînes) » à la page 264
escape (fonction)	« escape » à la page 264
ESCAPE (constante)	« Key.ESCAPE » à la page 286
eval	« eval » à la page 265
evaluate	« evaluate » à la page 266
exp	« Math.exp » à la page 302

Élément ActionScript	Voir entrée
firstChild	« XML.firstChild » à la page 399
floor	« Math.floor » à la page 302
_focusrect	« _focusrect » à la page 266
for	« for » à la page 267
for.. in	« for..in » à la page 268
_framesloaded	« _framesloaded » à la page 269
fromCharCode	« String.fromCharCode » à la page 374
fscommand	« fscommand » à la page 270
function	« function » à la page 271
ge	« ge (supérieur ou égal à–propre aux chaînes) » à la page 272
getAscii	« Key.getAscii » à la page 286
getBeginIndex	« Selection.getBeginIndex » à la page 354
getBounds	« MovieClip.getBounds » à la page 315
getBytesLoaded	« MovieClip.getBytesLoaded » à la page 316
getBytesTotal	« MovieClip.getBytesTotal » à la page 316
getCaretIndex	« Selection.getCaretIndex » à la page 354
getCode	« Key.getCode » à la page 287
getDate	« Date.getDate » à la page 245
getDay	« Date.getDay » à la page 246
getEndIndex	« Selection.getEndIndex » à la page 355
getFocus	« Selection.getFocus » à la page 355
getFullYear	« Date.getFullYear » à la page 246
getHours	« Date.getHours » à la page 247
getMilliseconds	« Date.getMilliseconds » à la page 247
getMinutes	« Date.getMinutes » à la page 247
getMonth	« Date.getMonth » à la page 248
getPan	« Sound.getPan » à la page 360
getProperty	« getProperty » à la page 273

Élément ActionScript	Voir entrée
getRGB	« Color.setRGB » à la page 239
getSeconds	« Date.getSeconds » à la page 248
getTime	« Date.getTime » à la page 248
getTimer	« getTimer » à la page 273
getTimezoneOffset	« Date.getTimezoneOffset » à la page 249
getTransform	« Color.getTransform » à la page 238, « Sound.getTransform » à la page 361
getURL	« getURL » à la page 274, « MovieClip.getURL » à la page 317
getUTCDate	« Date.getUTCDate » à la page 249
getUTCDay	« Date.getUTCDay » à la page 249
getUTCFullYear	« Date.getUTCFullYear » à la page 250
getUTCHours	« Date.getUTCHours » à la page 250
getUTCMilliseconds	« Date.getUTCMilliseconds » à la page 250
getUTCMinutes	« Date.getUTCMinutes » à la page 251
getUTCMonth	« Date.getUTCMonth » à la page 251
getUTCSeconds	« Date.getUTCSeconds » à la page 251
getVersion	« getVersion » à la page 275
getVolume	« Sound.getVolume » à la page 361
getYear	« Date.getYear » à la page 252
globalToLocal	« MovieClip.globalToLocal » à la page 317
gotoAndPlay	« gotoAndPlay » à la page 276, « MovieClip.gotoAndPlay » à la page 318
gotoAndStop	« gotoAndStop » à la page 276, « MovieClip.gotoAndStop » à la page 318
gt	« gt (supérieur à—propre aux chaînes) » à la page 277
hasChildNodes	« XML.haschildNodes » à la page 400
_height	« _height » à la page 277
hide	« Mouse.hide » à la page 311
_highquality	« _highquality » à la page 278

Élément ActionScript	Voir entrée
hitTest	« MovieClip.hitTest » à la page 319
HOME	« Key.HOME » à la page 287
if	« if » à la page 278
ifFrameLoaded	« ifFrameLoaded » à la page 279
#include	« #include » à la page 279
indexOf	« String.indexOf » à la page 375
Infinity	« Infinity » à la page 280
INSERT	« Key.INSERT » à la page 287
insertBefore	« XML.insertBefore » à la page 400
int	« int » à la page 280
isDown	« Key.isDown » à la page 288
isFinite	« isFinite » à la page 280
isNaN	« isNaN » à la page 281
isToggled	« Key.isToggled » à la page 288
join	« Array.join » à la page 226
Key	« Key (objet) » à la page 282
lastChild	« XML.lastChild » à la page 401
lastIndexOf	« String.indexOf » à la page 375
le	« le (inférieur ou égal à—propre aux chaînes) » à la page 291
LEFT	« Key.LEFT » à la page 288
length	« length » à la page 291, « Array.length » à la page 227, « String.length » à la page 375
LN2	« Math.LN2 » à la page 304
LN10	« Math.LN10 » à la page 304
load	« XML.load » à la page 401
loaded	« XML.loaded » à la page 402
loadMovie	« loadMovie » à la page 293, « MovieClip.loadMovie » à la page 320

Élément ActionScript	Voir entrée
loadVariables	« loadVariables » à la page 295, « MovieClip.loadVariables » à la page 321
localToGlobal	« MovieClip.localToGlobal » à la page 322
log	« Math.log » à la page 302
LOG2E	« Math.LOG2E » à la page 303
LOG10E	« Math.LOG10E » à la page 303
lt	« le (inférieur ou égal à—propre aux chaînes) » à la page 291
Math	« Math (objet) » à la page 296
max	« Math.max » à la page 304
maxscroll	« maxscroll » à la page 309
MAX_VALUE	« Number.MAX_VALUE » à la page 333
mbchr	« mbchr » à la page 309
mblength	« mblength » à la page 310
mbord	« mbord » à la page 310
mbsubstring	« mbsubstring » à la page 310
min	« Math.min » à la page 305
MIN_VALUE	« Number.MIN_VALUE » à la page 334
Mouse	« Mouse (objet) » à la page 311
MovieClip	« MovieClip (objet) » à la page 312
_name	« _name » à la page 326
NaN	« NaN » à la page 327, « Number.NaN » à la page 334
ne	« ne (inégalité—propre aux chaînes) » à la page 327
NEGATIVE_INFINITY	« Number.NEGATIVE_INFINITY » à la page 334
new **(opérateur)**	« new » à la page 328
newline	« newline » à la page 329
nextFrame	« nextFrame » à la page 329, « MovieClip.nextFrame » à la page 323
nextScene	« nextScene » à la page 329
nextSibling	« XML.nextSibling » à la page 403

Élément ActionScript	Voir entrée
nodeName	« XML.nodeName » à la page 403
nodeType	« XML.nodeType » à la page 404
nodeValue	« XML.nodeValue » à la page 404
not	« not » à la page 330
null	« null » à la page 330
Nombre	« Number (fonction) » à la page 331, « Nombre (objet) » à la page 332
Objet	« Objet (objet) » à la page 336
On	« on(mouseEvent) » à la page 340
onClipEvent	« onClipEvent » à la page 338
onClose	« XMLSocket.onClose » à la page 414
onConnect	« XMLSocket.onConnect » à la page 415
OnLoad	« XML.onLoad » à la page 404
onXML	« XMLSocket.onXML » à la page 416
or (OR logique)	« or » à la page 341
ord	« ord » à la page 342
_parent	« _parent » à la page 342
parentNode	« XML.parentNode » à la page 405
parseFloat	« parseFloat » à la page 343
parseInt	« parseInt » à la page 343
parseXML	« XML.parseXML » à la page 406
PGDN	« Key.PGDN » à la page 289
PGUP	« Key.PGUP » à la page 289
PI	« Math.PI » à la page 305
play	« play » à la page 344, « MovieClip.play » à la page 323
pop	« Array.pop » à la page 228
POSITIVE_INFINITY	« Number.POSITIVE_INFINITY » à la page 335
pow	« Math.pow » à la page 306

Élément ActionScript	Voir entrée
prevFrame	« prevFrame » à la page 345, « MovieClip.prevFrame » à la page 323
previousSibling	« XML.previousSibling » à la page 406
prevScene	« prevScene » à la page 346
print	« print » à la page 346
printAsBitmap	« printAsBitmap » à la page 348
push	« Array.push » à la page 228
_quality	« _quality » à la page 349
random	« random » à la page 350, « Math.random » à la page 306
removeMovieClip	« removeMovieClip » à la page 350, « MovieClip.removeMovieClip » à la page 324
removeNode	« XML.removeNode » à la page 407
return	« return » à la page 351
reverse	« Array.reverse » à la page 229
RIGHT	« Key.RIGHT » à la page 289
_root	« _root » à la page 351
_rotation	« _rotation » à la page 352
round	« Math.round » à la page 306
scroll	« scroll » à la page 353
Selection	« Selection (objet) » à la page 353
send	« XML.send » à la page 407, « XMLSocket.send » à la page 417
sendAndLoad	« XML.sendAndLoad » à la page 407
set	« set » à la page 356
setDate	« Date.setDate » à la page 252
setFocus	« Selection.setFocus » à la page 356
setFullYear	« Date.setFullYear » à la page 252
setHours	« Date.setHours » à la page 253
setMilliseconds	« Date.setMilliseconds » à la page 253

Élément ActionScript	Voir entrée
setMinutes	« Date.setMinutes » à la page 254
setMonth	« Date.setMonth » à la page 254
setPan	« Sound.setPan » à la page 361
setProperty	« setProperty » à la page 358
setRGB	« Color.setRGB » à la page 239
setSeconds	« Date.setSeconds » à la page 254
setSelection	« Selection.setSelection » à la page 356
setTime	« Date.setTime » à la page 255
setTransform	« Color.setTransform » à la page 239, « Sound.setTransform » à la page 363
setUTCDate	« Date.setUTCDate » à la page 255
setUTCFullYear	« Date.setUTCFullYear » à la page 255
setUTCHours	« Date.setUTCHours » à la page 256
setUTCMilliseconds	« Date.setUTCMilliseconds » à la page 256
setUTCMinutes	« Date.setUTCMinutes » à la page 257
setUTCMonth	« Date.setUTCMonth » à la page 257
setUTCSeconds	« Date.setUTCSeconds » à la page 258
setVolume	« Sound.setVolume » à la page 365
setYear	« Date.setYear » à la page 258
shift (méthode)	« Array.shift » à la page 229
SHIFT (constante)	« Key.SHIFT » à la page 290
show	« Mouse.show » à la page 312
sin	« Math.sin » à la page 307
slice	« Array.slice » à la page 230, « String.slice » à la page 376
sort	« Array.sort » à la page 230
Sound	« Son (objet) » à la page 358
_soundbuftime	« _soundbuftime » à la page 367
SPACE	« Key.SPACE » à la page 290

Élément ActionScript	Voir entrée
splice	« Array.splice » à la page 232
split	« String.split » à la page 376
sqrt	« Math.sqrt » à la page 307
SQRT1_2	« Math.SQRT1_2 » à la page 308
SQRT2	« Math.SQRT2 » à la page 308
start	« Sound.start » à la page 366
startDrag	« startDrag » à la page 368, « MovieClip.startDrag » à la page 324
status	« XML.status » à la page 408
stop	« stop » à la page 368, « MovieClip.stop » à la page 325, « Sound.stop » à la page 367
stopAllSounds	« stopAllSounds » à la page 369
stopDrag	« stopDrag » à la page 369, « MovieClip.stopDrag » à la page 325
Chaîne	« String (fonction) » à la page 370, « Chaîne (objet) » à la page 371, « " " (délimiteur de chaîne) » à la page 371
substr	« String.substr » à la page 377
substring	« substring » à la page 378, « String.substring » à la page 377
swapDepths	« MovieClip.swapDepths » à la page 325
TAB	« Key.TAB » à la page 290
tan	« Math.tan » à la page 308
_target	« _target » à la page 379
targetPath	« targetPath » à la page 379
tellTarget	« tellTarget » à la page 380
this	« this » à la page 381
toggleHighQuality	« toggleHighQuality » à la page 382
toLowerCase	« String.toLowerCase » à la page 378
toString	« Array.toString » à la page 232, « Boolean.toString » à la page 235, « Date.toString » à la page 258, « Number.toString » à la page 335, « Object.toString » à la page 337, « XML.toString » à la page 409

Élément ActionScript	Voir entrée
_totalframes	« _totalframes » à la page 382
toUpperCase	« String.toUpperCase » à la page 378
trace	« trace » à la page 383
typeof	« typeof » à la page 384
unescape	« unescape » à la page 384
unloadMovie	« unloadMovie » à la page 385, « MovieClip.unloadMovie » à la page 326
unshift	« Array.shift » à la page 229
UP	« Key.UP » à la page 291
updateAfterEvent	« updateAfterEvent » à la page 385
_url	« _url » à la page 386
UTC	« Date.UTC » à la page 259
valueOf	« Boolean.valueOf » à la page 235, « Number.valueOf » à la page 336, « Object.valueOf » à la page 337
var	« var » à la page 387
_visible	« _visible » à la page 387
void	« void » à la page 388
while	« while » à la page 388
_width	« _width » à la page 389
with	« with » à la page 390
_x	« _x » à la page 393
XML	« XML (objet) » à la page 393
xmlDecl	« XML.xmlDecl » à la page 409
XMLSocket	« XMLSocket (objet) » à la page 410
_xmouse	« _xmouse » à la page 418
_xscale	« _xscale » à la page 418
_y	« _y » à la page 419
_ymouse	« _ymouse » à la page 420
_yscale	« _yscale » à la page 420

-- (décrémentation)

Syntaxe

```
--expression
expression--
```

Arguments

expression Une variable, un nombre, l'élément d'un tableau ou la propriété d'un objet.

Description

Opérateur ; un opérateur unaire de pré-décrémentation et de post-décrémentation qui soustrait 1 à l'*expression*. La forme de pré-décrémentation de l'opérateur (*--expression*) soustrait 1 à l'*expression* et renvoie le résultat. La forme de post-décrémentation de l'opérateur (*expression--*) soustrait 1 à l'*expression* et renvoie la valeur initiale de l'*expression* (le résultat avant la soustraction).

Lecteur

Flash 4 et versions suivantes.

Exemple

La forme de pré-décrémentation de l'opérateur décrémente `x` de 2 (`x - 1 = 2`) et renvoie le résultat sous la forme `y` :

```
x = 3;
y = --x
```

La forme de post-décrémentation de l'opérateur décrémente `x` de 2 (`x - 1 = 2`) et renvoie la valeur originale (`x = 3`) comme résultat `y` :

```
If x = 3;
y = x--
```

++ (incrémentation)

Syntaxe

```
++expression
expression++
```

Arguments

expression Une variable, un nombre, l'élément d'un tableau ou la propriété d'un objet.

Description

Opérateur ; un opérateur unaire de pré-incrémentation et de post-incrémentation qui ajoute 1 à l'*expression*. La forme de pré-incrémentation de l'opérateur (*++expression*) ajoute 1 à l'*expression* et renvoie le résultat. La forme de post-incrémentation de l'opérateur (*expression++*) ajoute 1 à l'*expression* et renvoie la valeur initiale de l'*expression* (le résultat avant l'addition).

La forme de pré-incrémentation de l'opérateur incrémente `x` de 2 (`x + 1 = 2`) et renvoie le résultat sous la forme `y` :

```
x = 1;
y = ++x
```

La forme de post-incrémentation de l'opérateur incrémente `x` de 2 (`x + 1 = 2`) et renvoie la valeur d'origine (`x = 1`) comme résultat `y` :

```
x = 1;
y = x++
```

Lecteur

Flash 4 et versions suivantes.

Exemple

L'exemple suivant utilise ++ comme opérateur de pré-incrémentation avec une instruction `while`.

```
i = 0
while(i++ < 5){
// this section will execute five times
}
```

L'exemple suivant utilise ++ comme opérateur de pré-incrémentation :

```
var a = [];
var i = 0;
while (i < 10) {
   a.push(++i);
}
trace(a.join());
```

Ce script écrit la séquence suivante :

```
1,2,3,4,5,6,7,8,9
```

L'exemple suivant utilise ++ comme opérateur de post-incrémentation :

```
var a = [];
var i = 0;
while (i < 10) {
a.push(i++);
   }
trace(a.join());
```

Ce script écrit la séquence suivante :

```
0,1,2,3,4,5,6,7,8,9
```

! (NOT logique)

Syntaxe

`!expression`

Arguments

`expression` Une variable ou une expression évaluée.

Description

Opérateur (logique) ; inverse la valeur booléenne d'une variable ou d'une expression. Si *`expression`* est une variable avec une valeur absolue ou convertie `true`, `!variable`, la valeur de `! expression` est `false`. Si l'expression `x && y` est `false`, l'expression `!(x && y)` est `true`. Cet opérateur est identique à l'opérateur `not` qui était utilisé dans Flash 4.

Lecteur

Flash 4 et versions suivantes.

Exemple

Dans l'exemple suivant, la variable `happy` est définie sur `false`, la condition `if` évalue la condition `!happy`, et si la condition est `true`, `trace` envoie une chaîne dans la fenêtre Résultat.

```
happy = false;
if (!happy){
trace("don't worry be happy");
}
```

L'exemple suivant illustre le résultat de l'opérateur `!` :

`!` *`true`* renvoie `false`

`!` *`false`* renvoie `true`

!= (inégalité)

Syntaxe

`expression1` `!=` *`expression2`*

Arguments

`expression1`, *`expression2`* Nombres, chaînes, booléens, variables, objets, tableaux ou fonctions.

Description

Opérateur (égalité) ; teste l'opposé exact de l'opérateur ==. Si *`expression1`* est égale à *`expression2`*, le résultat est `false`. Comme pour l'opérateur ==, la définition de *égale* dépend des types de données comparés.

- Les nombres, les chaînes et les valeurs booléennes sont comparés par valeur.
- Les variables, les objets, les tableaux et les fonctions sont comparés par référence.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant illustre les résultats de l'opérateur `!=`.

`5 != 8` renvoie `true`

`5 != 5` renvoie `false`

L'exemple suivant illustre l'utilisation de l'opérateur `!=` dans une instruction `if` :

```
a = "David";
b = "Fool"
if (a != b)
trace("David is not a fool");
```

Voir aussi
« == (égalité) » à la page 216

% (pourcentage)

Syntaxe
expression1 % expression2

Arguments
expression1, expression2 Nombres, entiers, nombres à virgule flottante ou chaînes se convertissant en valeurs numériques.

Description
Opérateur (arithmétique) ; calcule le reste de *expression1* divisé par *expression2*. Si l'un des arguments de *expression* n'est pas un nombre, l'opérateur de pourcentage tente de le convertir en nombre.

Lecteur
Flash 4 et versions suivantes. Dans les fichiers Flash 4, l'opérateur `%` est développé dans le fichier SWF sous la forme `x - int(x/y) * y` et n'est pas aussi rapide, ni aussi précis, que l'implémentation Flash 5 Player.

Exemple
L'exemple suivant montre l'emploi numérique de l'opérateur `%` :

`12 % 5` renvoie `2`

`4.3 % 2.1` renvoie `0.1`

%= (affectation de pourcentage)

Syntaxe
expression1 %= *expression2*

Arguments
expression1, *expression2* Entiers et variables.

Description
Opérateur (affectation) ; affecte à *expression1* la valeur de *expression1* % *expression2*.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant illustre l'utilisation de l'opérateur %= avec des variables et des nombres :

x %= y est identique à x = x % y

Si x = 14 et y = 5 alors

x %= 5 renvoie 4

Voir aussi
« % (pourcentage) » à la page 192

& (AND au niveau du bit)

Syntaxe
expression1 & *expression2*

Arguments
expression1, *expression2* Tout nombre.

Description
Opérateur (au niveau du bit) ; convertit *expression1* et *expression2* en entiers de 32 bits non signés et effectue une opération AND booléenne sur chaque bit des arguments entiers. Le résultat est un nouvel entier non signé de 32 bits.

Lecteur
Flash 5 et versions suivantes. Dans Flash 4, l'opérateur & était utilisé pour concaténer les chaînes. Dans Flash 5, l'opérateur & est un AND au niveau du bit et les opérateurs add et + concatènent les chaînes. Les fichiers Flash 4 utilisant l'opérateur & sont automatiquement mis à jour pour utiliser add lorsqu'ils sont importés dans l'environnement de programmation de Flash 5.

&& (AND court-circuit)

Syntaxe
expression1 && *expression2*

Arguments
expression1, expression2 Nombres, chaînes, variables ou fonctions.

Description
Opérateur (logique) ; effectue une opération booléenne sur les valeurs d'une ou des deux expressions. Oblige l'interprète de Flash à évaluer *expression1* (l'expression de gauche) et renvoie `false` si l'expression évalue `false`. Si *expression1* évalue `true`, *expression2* (à droite) est évaluée. Si *expression2* évalue `true`, le résultat final est `true` ; sinon, il est `false`.

Lecteur
Flash 4 et versions suivantes.

Exemple
Cet exemple affecte les valeurs des expressions évaluées aux variables `winner` et `loser` afin d'effectuer un test :

```
winner = (chocolateEggs >=10) && (jellyBeans >=25);
loser = (chocolateEggs <=1) && (jellyBeans <= 5);
if (winner) {
   alert = "You Win the Hunt!";
   if (loser) {
      alert = "Now THAT'S Unhappy Hunting!";
   }
} else {
   alert = "We're all winners!";
}
```

&= (affectation AND au niveau du bit)

Syntaxe
expression1 &= *expression2*

Arguments
expression1, expression2 Entiers et variables.

Description
Opérateur (affectation au niveau du bit) ; affecte à *expression1* la valeur de *expression1* & *expression2*.

Lecteur
Flash 5 et versions suivantes.

Exemple

L'exemple suivant illustre l'utilisation de l'opérateur `&=` avec des variables et des nombres :

`x &= y` est identique à `x = x & y`

Si `x = 15` et `y = 9` alors

`x &= 9` renvoie `9`

Voir aussi

« & (AND au niveau du bit) » à la page 193

() (parenthèses)

Syntaxe

```
(expression1, expression2);
function(functionCall1, ..., functionCallN);
```

Arguments

expression1, expression2 Nombres, chaînes, variables ou texte.

function La fonction devant être effectuée sur le contenu des parenthèses.

functionCall1...functionCallN Une série de fonctions à exécuter avant que le résultat ne soit transmis à la fonction extérieure aux parenthèses.

Description

Opérateur (général) ; effectue une opération de groupement sur un ou plusieurs arguments ou entoure un ou plusieurs arguments et transmet le résultat sous forme de paramètres à une fonction extérieure aux parenthèses.

Emploi 1 : Effectue une opération de groupement sur une ou plusieurs expressions pour contrôler l'ordre d'exécution des opérateurs dans l'expression. Cet opérateur supplante l'ordre de priorité automatique et oblige les expressions entre parenthèses à être évaluées en premier. Lorsque les parenthèses sont imbriquées, Flash évalue le contenu des parenthèses internes avant le contenu des parenthèses externes.

Emploi 2 : Entoure un ou plusieurs arguments et les transmet sous forme de paramètres à la fonction extérieure aux parenthèses.

Lecteur

Flash 4 et versions suivantes.

Exemple
(Emploi 1) Les instructions suivantes illustrent l'emploi des parenthèses pour contrôler l'ordre d'exécution des expressions. (Le résultat apparaît sous chaque instruction.)

```
(2 + 3) * (4 + 5)
45
2 + (3 * (4 + 5))
29
2 + (3 * 4) + 5
19
```

(Emploi 2) L'exemple suivant illustre l'emploi des parenthèses avec une fonction :

```
getDate();
invoice(item, amount);
```

Voir aussi
« with » à la page 390

- (moins)

Syntaxe
(Négation) –*expression*

(Soustraction) *expression1* - *expression2*

Arguments
expression1, expression2 Tout nombre.

Description
Opérateur (arithmétique) ; utilisé pour la négation ou la soustraction. Lorsqu'il est utilisé pour la négation, il inverse le signe de l'*expression* numérique. Lorsqu'il est utilisé pour la soustraction, il effectue une soustraction arithmétique sur deux expressions numériques, soustrayant *expression2* de *expression1*. Lorsque les deux expressions sont des entiers, la différence est un entier. Lorsque l'une des expressions, ou les deux, est un nombre à virgule flottante, la différence est un nombre à virgule flottante.

Lecteur
Flash 4 et versions suivantes.

Exemple
(Négation) Cette instruction inverse le signe de l'expression 2 + 3 :

```
-(2 + 3)
```

Le résultat est -5.

(Soustraction) Cette instruction soustrait l'entier 2 de l'entier 5 :

```
5 - 2
```

Le résultat est 3, qui est un entier.

(Soustraction) : Cette instruction soustrait le nombre à virgule flottante 1,5 du nombre à virgule flottante 3,25 :

```
put 3.25 - 1.5
```

Le résultat est 1,75, qui est un nombre à virgule flottante.

* (multiplication)

Syntaxe
*expression1 * expression2*

Arguments
expression1, expression2 Entiers ou nombres à virgule flottante.

Description
Opérateur (arithmétique) ; multiplie deux expressions numériques. Si les deux expressions sont des entiers, le produit est un entier. Si l'une des expressions, ou les deux, est un nombre à virgule flottante, le produit est un nombre à virgule flottante.

Lecteur
Flash 4 et versions suivantes.

Exemple
Cette instruction multiplie les entiers 2 et 3 :

```
2 * 3
```

Le résultat est 6, qui est un entier.

Exemple
Cette instruction multiplie les nombres à virgule flottante 2,0 et 3,1416 :

```
2.0 * 3.1416
```

Le résultat est 6,2832, qui est un nombre à virgule flottante.

*= (affectation de multiplication)

Syntaxe
*expression1 *= expression2*

Arguments
expression1, expression2 Entiers, nombres à virgule flottante ou chaînes.

Description
Opérateur (affectation) ; affecte à *expression1* la valeur de *expression1 * expression2*.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant illustre l'utilisation de l'opérateur *= avec des variables et des nombres :

`x *= y` est identique à `x = x * y`

Si `x = 5` et `y = 10` alors

`x *= 10` renvoie `50`

Voir aussi
« * (multiplication) » à la page 197

, (virgule)

Syntaxe
expression1, expression2

Arguments
expression Un nombre, une variable, une chaîne, un élément de tableau ou une autre donnée.

Description
Opérateur ; indique à Flash d'évaluer *expression1*, puis *expression2* et renvoie la valeur de *expression2*. Cet opérateur est principalement utilisé avec l'instruction de boucle `for`.

Lecteur
Flash 4 et versions suivantes.

Exemple

L'exemple de code suivant utilise l'opérateur virgule :

```
var a=1, b=2, c=3;
```

Ce qui revient à écrire la séquence suivante :

```
var a=1;
var b=2;
var c=3;
```

. (opérateur point)

Syntaxe

```
object.property_or_method
instancename.variable
instancename.childinstance.variable
```

Arguments

object Une occurrence d'un objet. Certains objets nécessitent que les occurrences soient créées en utilisant le constructeur de cet objet. L'objet peut être un des objets prédéfinis ActionScript ou un objet personnalisé. Cet argument se situe toujours à gauche de l'opérateur point (.).

property_or_method Le nom d'une propriété ou d'une méthode associée à l'objet. Toutes les méthodes et propriétés valides pour les objets prédéfinis sont répertoriées dans les tableaux résumant les propriétés et méthodes de cet objet. Cet argument se situe toujours à droite de l'opérateur point (.).

instancename Le nom d'une occurrence de clip.

childinstance Une occurrence de clip qui est un enfant du clip principal.

variable Une variable dans un clip.

Description

Opérateur ; utilisé pour naviguer dans la hiérarchie du clip afin d'accéder aux clips enfants imbriqués, aux variables ou aux propriétés. L'opérateur point est également utilisé pour tester ou définir les propriétés d'un objet, exécuter une méthode d'un objet ou créer une structure de données.

Lecteur

Flash 4 et versions suivantes.

Voir aussi

« [] (opérateur accès tableau) » à la page 203

Exemple
Cette instruction identifie la valeur courante de la variable `hairColor` par le clip `person` :

```
person.hairColor
```

Ceci est identique à la syntaxe de Flash 4 suivante :

```
/person:hairColor
```

Exemple
Le code suivant illustre l'utilisation de l'opérateur point pour créer une structure d'un tableau :

```
account.name = "Gary Smith";
account.address = "123 Main St ";
account.city = "Any Town";
account.state = "CA";
account.zip = "12345";
```

?: (conditionnel)

Syntaxe
expression1 ? *expression2* : *expression3*

Arguments
expression1 Une expression qui évalue en valeur booléenne, généralement une expression de comparaison.

expression2, expression3 Valeurs de tous types.

Description
Opérateur (conditionnel) ; indique à Flash d'évaluer *expression1* et renvoie la valeur de *expression2* si *expression1* est `true` ; sinon, renvoie la valeur de *expression3*.

Lecteur
Flash 4 et versions suivantes.

/ (division)

Syntaxe
expression1 / *expression2*

Arguments
expression Tout nombre.

Description
Opérateur (arithmétique) ; divise *expression1* par *expression2*. Les arguments de l'expression et les résultats de l'opération de division sont traités comme des nombres à virgule flottante double précision.

Lecteur
Flash 4 et versions suivantes.

Exemple
Cette instruction divise le nombre à virgule flottante 22,0 par 7,0 et affiche ensuite le résultat dans la fenêtre Résultat :

```
trace(22.0 / 7.0);
```

Le résultat est 3,1429, qui est un nombre à virgule flottante.

// (délimiteur de commentaires)

Syntaxe
```
// commentaire
```

Arguments
commentaire Texte qui ne fait pas partie du code et qui doit être ignoré par l'interpréteur.

Description
Commentaire ; indique le début d'un commentaire de script. Tout texte qui apparaît entre le délimiteur de commentaires `//` et le caractère de fin de ligne est interprété comme un commentaire et ignoré par l'interpréteur ActionScript.

Lecteur
Flash 1 et versions suivantes.

Exemple
Ce script utilise les barres obliques qui délimitent les commentaires pour identifier les première, troisième, cinquième et septième lignes comme commentaires :

```
// set the X position of the ball movie clip
ball = getProperty(ball._x);
// set the Y position of the ball movie clip
ball = getProperty(ball._y);
// set the X position of the kitty movie clip
kitty = getProperty(kitty._x);
// set the Y position of the kitty movie clip
kitty_y = getProperty(kitty._y);
```

Voir aussi
« /* (délimiteur de commentaires) » à la page 202

/* (délimiteur de commentaires)

Syntaxe

```
/* commentaire */
/*
* commentaire
* commentaire
*/
```

Arguments

commentaire Tout texte.

Description

Commentaire ; indique une ou plusieurs lignes de commentaires de script. Tout texte qui apparaît entre la balise d'ouverture de commentaires `/*` et la balise de fermeture de commentaires `*/` est interprété comme un commentaire et ignoré par l'interpréteur ActionScript. Utilisez la première syntaxe pour identifier un commentaire d'une seule ligne et utilisez la seconde syntaxe pour identifier un commentaire sur plusieurs lignes successives. L'oubli de la balise de fermeture `*/` lors de l'utilisation de cette forme de délimiteur de commentaires provoque un message d'erreur envoyé par le compilateur ActionScript.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« // (délimiteur de commentaires) » à la page 201

/= (affectation de division)

Syntaxe

```
expression1 /= expression2
```

Arguments

expression1, expression2 Entiers, nombres à virgule flottante ou chaînes.

Description

Opérateur (affectation) ; affecte à *expression1* la valeur de *expression1* / *expression2*.

Lecteur

Flash 4 et versions suivantes.

Exemple

L'exemple suivant illustre l'utilisation de l'opérateur `/=` avec des variables et des nombres :

```
x /= y is the same as x = x /y
x = 10;
y = 2;
x /= y;
// x now contains the value 5
```

[] (opérateur accès tableau)

Syntaxe

```
myArray["a0", "a1",..."aN"];
```

```
object[value1, value2, ...valueN];
```

Arguments

myArray Le nom d'un tableau.

a0, a1,...aN Les éléments d'un tableau.

value1, 2,...N Les noms de propriétés.

Description

Opérateur ; crée un nouvel objet initialisant les propriétés spécifiées dans les arguments ou initialise un nouveau tableau avec les éléments (*a0*) spécifiés dans les arguments.

L'objet créé possède comme prototype l'objet générique `Objet`. L'utilisation de cet opérateur est identique à l'utilisation de l'opérateur d'affectation pour appeler `new Object` et attribuer les propriétés. L'utilisation de cet opérateur est une alternative à l'utilisation de l'opérateur `new` qui permet la création rapide et pratique d'objets.

Lecteur

Flash 4 et versions suivantes.

Exemple

Les aperçus d'exemples de codes suivants sont deux manières différentes de créer un nouvel objet Chaîne vide.

```
myArray =[];
myArray = new Array();
```

L'exemple suivant montre un tableau simple.

```
myArray = ["red", "orange", "yellow", "green", "blue", "purple"]
myArray[0]="red"
myArray[1]="yellow"
myArray[2]="green"
myArray[3]="blue"
myArray[4]="purple"
```

^(XOR au niveau du bit)

Syntaxe

expression1 ^ *expression2*

Arguments

expression1, expression2 Tout nombre.

Description

Opérateur (au niveau du bit) ; convertit *expression1* et *expression2* en entiers de 32 bits non signés et renvoie un 1 dans chaque position de bit où les bits correspondants dans *expression1* ou *expression2*, mais pas les deux, sont 1.

Lecteur

Flash 5 et versions suivantes.

Exemple

```
15 ^ 9 returns 6
(1111 ^ 1001 = 0110)
```

^= (affectation XOR au niveau du bit)

Syntaxe

expression1 ^= *expression2*

Arguments

expression1, expression2 Entiers et variables.

Description

Opérateur (affectation composée) ; affecte à *expression1* la valeur de *expression1* ^ *expression2*.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant montre une opération ^=.

```
// 15 decimal = 1111 binary
x = 15;
// 9 decimal = 1001 binary
x ^= y;
returns
x ^ y (0110 binary)
```

L'exemple suivant illustre l'utilisation de l'opérateur ^= avec des variables et des nombres :

```
x ^= y is the same as x = x ^ y
If x = 15 and y = 9 then
15 ^= 9 returns 6
```

Voir aussi

« ^(XOR au niveau du bit) » à la page 204

{} (initialisateur d'objet)

Syntaxe

```
object {name1: value1,
   name1: value2,
...
   nameN: valueN };
```

Arguments

object L'objet à créer.

name1,2,...N Le nom de la propriété.

value1,2,...N La valeur correspondant à chaque propriété *name*.

Description

Opérateur ; crée un nouvel objet et l'initialise avec les paires de propriétés spécifiées *name* et *value*. L'objet créé possède comme prototype l'objet générique `Object`. L'utilisation de cet opérateur est identique à l'utilisation de l'opérateur d'affectation pour appeler `new Object` et attribuer les paires de propriétés. L'utilisation de cet opérateur est une alternative à l'utilisation de l'opérateur `new` qui permet la création rapide et pratique d'objets.

Lecteur

Flash 5 et versions suivantes.

Exemple

Le code suivant montre comment créer un objet vide en utilisant l'opérateur initialisateur d'objet et `new Object`.

```
object = {};
object = new Object();
```

L'exemple suivant crée un objet `account (compte)` initialisant les propriétés `name`, `address`, `city`, `state`, `zip` et `balance`.

```
account = { name: "John Smith",
   address: "123 Main Street",
   city: "Blossomville",
   state: "California",
   zip: "12345",
   balance: "1000" };
```

L'exemple suivant montre comment les initialisateurs d'objet et de tableau peuvent être imbriqués les uns dans les autres.

```
person = { name: "Peter Piper",
   children: [ "Jack", "Jill", "Moe",] };
```

L'exemple suivant est une autre manière d'utiliser les informations de l'exemple précédent, avec les mêmes résultats.

```
person = new Person();
person.name = 'John Smith';
person.children = new Array();
person.children[0] = 'Jack';
person.children[1] = 'Jill';
person.children[2] = 'Moe';
```

Voir aussi

« [] (opérateur accès tableau) » à la page 203
« new » à la page 328
« Objet (objet) » à la page 336

| (OR au niveau du bit)

Syntaxe

expression1 | *expression2*

Arguments

expression1, expression2 Tout nombre.

Description

Opérateur (au niveau du bit) ; convertit *expression1* et *expression2* en entiers de 32 bits non signés et renvoie un 1 à chaque position de bit où les bits correspondants de *expression1* ou de *expression2* sont 1.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant montre une opération OR au niveau du bit. Remarquez que 15 s'écrit 1111 en binaire.

```
// 15 decimal = 1111 binary
x = 15;
// 9 decimal = 1001 binary
y = 9;
// x | y = binary
z = x | y;
z = 15
```

L'exemple suivant montre une façon différente d'exprimer l'exemple précédent.

```
15 | 9 returns 15
(1111 | 0011 = 1111)
```

|| (OR)

Syntaxe

expression1 || *expression2*

Arguments

expression1, expression2 Une valeur booléenne ou une expression qui convertit en une valeur booléenne.

Description

Opérateur (logique) ; évalue *expression1* et *expression2*. Le résultat est (`true`) si une des deux ou les deux expressions évaluent `true` ; le résultat est (`false`) seulement si les deux expressions évaluent `false`.

Avec des expressions non booléennes, l'opérateur logique OR oblige Flash à évaluer l'expression de gauche ; si elle peut être convertie en `true`, le résultat est `true`. Sinon, Flash évalue l'expression de droite et le résultat est la valeur de cette expression.

Lecteur

Flash 4 et versions suivantes.

Exemple

L'exemple suivant utilise l'opérateur `||` dans une instruction `if` :

```
want = true;
need = true;
love = false;
if (want || need || love){
trace("two out of 3 ain't bad");
}
```

|= (affectation OR au niveau du bit)

Syntaxe

expression1 |= *expression2*

Arguments

expression1, expression2 Entiers et variables.

Description

Opérateur (affectation) ; affecte à *expression1* la valeur de *expression1* | *expression2*.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant illustre l'utilisation de l'opérateur |= avec des variables et des nombres :

```
x |= y is the same as x = x | y
If x = 15 and y = 9 then
x |= 9 returns 15
```

Voir aussi

« | (OR au niveau du bit) » à la page 206

~ (NOT au niveau du bit)

Syntaxe

~ *expression*

Arguments

expression Tout nombre.

Description

Opérateur (au niveau du bit) ; convertit l'*expression* en entier de 32 bits non signé, puis inverse les bits. Ou, plus simplement, change le signe d'un nombre et soustrait 1.

Une opération NOT au niveau du bit modifie le signe d'un nombre et soustrait 1.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant est une explication numérique d'une opération NOT au niveau du bit effectuée sur une variable :

```
~a, returns -1 if a = 0, and returns -2 if a = 1, thus:
~0=-1 and ~1=-2
```

+ (addition)

Syntaxe

expression1 + expression2

Arguments

expression1, expression2 Entiers, nombres, nombres à virgule flottante ou chaînes.

Description

Opérateur ; additionne des expressions numériques ou concatène des chaînes. Si une expression est une chaîne, toutes les autres expressions sont converties en chaînes et concaténées.

Si les deux expressions sont des entiers, la somme est un entier ; si une des deux ou les deux expressions sont des nombres à virgule flottante, la somme est un nombre à virgule flottante.

Lecteur

Flash 4, Flash 5 et versions ultérieures. Dans Flash 5, + est un opérateur numérique ou un concaténateur de chaînes, selon le type de données de l'argument. Dans Flash 4, + est seulement un opérateur numérique. Les fichiers Flash 4 importés dans l'environnement de programmation de Flash 5 subissent un processus de conversion pour préserver l'intégrité des types de données. Le premier exemple ci-dessous illustre le processus de conversion.

Exemple

L'exemple suivant illustre la conversion d'un fichier Flash 4 contenant une comparaison d'égalité numérique.

Fichier Flash 4 :

```
x + y
```

Fichier Flash 5 converti :

```
Number(x) + Number(y)
```

Cette instruction additionne les entiers 2 et 3 et affiche ensuite le résultat, l'entier 5, dans la fenêtre Résultat :

```
trace (2 + 3);
```

Cette instruction additionne les nombres à virgule flottante 2,5 et 3,25 et affiche le résultat 5,7500, un nombre à virgule flottante, dans la fenêtre Résultat :

```
trace (2.5 + 3.25);
```

Cette instruction concatène deux chaînes et affiche le résultat, “today is my birthday” dans la fenêtre Résultat :

```
"today is my" + "birthday"
```

Voir aussi

« add » à la page 222

+= (affectation d'addition)

Syntaxe
expression1 += *expression2*

Arguments
expression1, expression2 Entiers, nombres à virgule flottante ou chaînes.

Description
Opérateur (affectation composée) ; affecte à *expression1* la valeur de *expression1* + *expression2*. Cette opérateur effectue également la concaténation de chaînes.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant illustre l'utilisation numérique de l'opérateur += :

```
x += y is the same as x = x + y
If x = 5 and y = 10 then
x += 10 returns 15
```

Cette exemple illustre l'utilisation de l'opérateur += avec une expression de chaîne :

```
x = "My name is"
x += "Mary"
```

Le résultat du code précédent sera :

```
"My name is Mary"
```

Voir aussi
« + (addition) » à la page 209

< (inférieur à)

Syntaxe
expression1 < *expression2*

Arguments
expression1, expression2 Nombres ou chaînes.

Description
Opérateur (comparaison) ; compare deux expressions et détermine si *expression1* est inférieure à *expression2* (`true`), ou si *expression1* est supérieure ou égale à *expression2* (`false`). Les expressions de chaînes sont évaluées et comparées selon le nombre de caractères dans la chaîne.

Lecteur

Flash 4 ; Flash 5 et versions suivantes. Dans Flash 5, < est un opérateur de comparaison capable de manipuler divers types de données. Dans Flash 4, < est un opérateur numérique. Les fichiers Flash 4 importés dans l'environnement de programmation de Flash 5 subissent un processus de conversion pour préserver l'intégrité des types de données. Le premier exemple ci-dessous illustre le processus de conversion.

Exemple

L'exemple suivant illustre la conversion d'un fichier Flash 4 contenant une comparaison d'égalité numérique.

Fichier Flash 4 :

```
x < y
```

Fichier Flash 5 converti :

```
Number(x) < Number(y)
```

Les exemples suivants illustrent les renvois `true` et `false` pour des nombres et de chaînes :

```
3 < 10 or "Al" < "Jack" return true
10 < 3 or "Jack" < "Al" return false
```

<< (décalage gauche au niveau du bit)

Syntaxe

expression1 << *expression2*

Arguments

expression1 Un nombre, une chaîne ou une expression devant être décalés à gauche.

expression2 Un nombre, une chaîne ou une expression convertis en entier de 0 à 31.

Description

Opérateur (au niveau du bit) ; convertit *expression1* et *expression2* en entiers de 32 bits et décale tous les bits de *expression1* vers la gauche du nombre de places spécifié par l'entier résultant de la conversion de *expression2*. Les emplacements des bits vidés par cette opération sont remplis par des 0. Décaler une valeur sur la gauche de 1 revient à la multiplier par deux.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant décale l'entier `1` de dix bits vers la gauche.

```
x = 1 << 10
```

Le résultat de cette opération est `x = 1024`. Ceci car 1 décimal égale 1 binaire, 1 binaire décalé sur la gauche de 10 est 10000000000 binaire, et 10000000000 binaire est 1024 décimal.

L'exemple suivant décale l'entier `7` de huit bits sur la gauche.

```
x = 7 << 8
```

Le résultat de cette opération est `x = 1792`. Ceci car 7 décimal égale 111 binaire, 111 binaire décalé sur la gauche de 8 bits est 11100000000 binaire, et 11100000000 binaire est 1792 décimal.

Voir aussi

« >>= (décalage droit au niveau du bit et affectation) » à la page 219

<<= (décalage gauche au niveau du bit et affectation)

Syntaxe

expression1 <<= *expression2*

Arguments

expression1 Un nombre, une chaîne ou une expression devant être décalés sur la gauche.

expression2 Un nombre, une chaîne ou une expression convertis en entier de 0 à 31.

Description

Opérateur (affectation composée) ; cet opérateur effectue une opération de décalage gauche au niveau du bit et stocke le contenu comme résultat dans *expression1*.

Lecteur

Flash 5 et versions suivantes.

Exemple

Les deux expressions suivantes sont identiques.

```
A <<= B
A = (A << B)
```

Voir aussi

« << (décalage gauche au niveau du bit) » à la page 211
« >>= (décalage droit au niveau du bit et affectation) » à la page 219

<= (inférieur ou égal à)

Syntaxe

expression1 <= *expression2*

Arguments

expression1, expression2 Nombres ou chaînes.

Description

Opérateur (comparaison) ; compare deux expressions et détermine si *expression1* est inférieure ou égale à *expression2* (true), ou si *expression1* est supérieure à *expression2* (false).

Lecteur

Flash 4, Flash 5 et versions ultérieures. Dans Flash 5, <= est un opérateur de comparaison capable de manipuler divers types de données. Dans Flash 4, <= est un opérateur numérique. Les fichiers Flash 4 importés dans l'environnement de programmation de Flash 5 subissent un processus de conversion pour préserver l'intégrité des types de données. Le premier exemple ci-dessous illustre le processus de conversion.

Exemple

L'exemple suivant illustre la conversion d'un fichier Flash 4 contenant une comparaison d'égalité numérique.

Fichier Flash 4 :

```
x <= y
```

Fichier Flash 5 converti :

```
Number(x) <= Number(y)
```

L'exemple suivant illustre les résultats true et false pour des nombres et des chaînes :

```
5 <= 10 or "Al" <= "Jack" returns true
10<= 5 or "Jack" <= "Al" returns false
```

<> (inégalité)

Syntaxe
expression1 <> *expression2*

Arguments
expression1, expression2 Nombres, chaînes, booléens, variables, objets, tableaux ou fonctions.

Description
Opérateur (égalité) ; teste l'opposé exact de l'opérateur ==. Si *expression1* est égale à *expression2*, le résultat est `false`. Tout comme pour l'opérateur ==, la définition de *égale* dépend des types de données comparés.

- Les nombres, les chaînes et les valeurs booléennes sont comparés par valeur.
- Les variables, les objets, les tableaux et les fonctions sont comparés par référence.

Cet opérateur a été désapprouvé dans Flash 5 et les utilisateurs sont encouragés à utiliser le nouvel opérateur !=.

Lecteur
Flash 2 et versions suivantes.

Voir aussi
« != (inégalité) » à la page 191

= (affectation)

Syntaxe
expression1 = *expression2*

Arguments
expression1 Une variable, l'élément d'un tableau ou la propriété d'un objet.

expression2 Une valeur de tout type.

Description
Opérateur (affectation) ; affecte le type de *expression2* (l'argument de droite) à la variable, à l'élément de tableau ou à la propriété dans *expression1*.

Lecteur
Flash 4, Flash 5 et versions ultérieures. Dans Flash 5, = est un opérateur d'affectation et l'opérateur == est utilisé pour évaluer une égalité. Dans Flash 4, = est un opérateur d'égalité numérique. Les fichiers Flash 4 importés dans l'environnement de programmation de Flash 5 subissent un processus de conversion pour préserver l'intégrité des types de données. Le premier exemple ci-dessous illustre le processus de conversion.

Exemple

L'exemple suivant illustre la conversion d'un fichier Flash 4 contenant une comparaison d'égalité numérique.

Fichier Flash 4 :

```
x = y
```

Fichier Flash 5 converti :

```
Number(x) == Number(y)
```

L'exemple suivant utilise l'opérateur d'affectation pour affecter le type de données number à la variable `x`.

```
x = 5
```

L'exemple suivant utilise l'opérateur d'affectation pour affecter le type de données string à la variable `x`.

```
x = "hello"
```

-= (affectation de négation)

Syntaxe

`expression1 -= expression2`

Arguments

`expression1, expression2` Entiers, nombres à virgule flottante ou chaînes.

Description

Opérateur (affectation composée) ; affecte à *`expression1`* la valeur de *`expression1 - expression2`*.

Lecteur

Flash 4 et versions suivantes.

Exemple

L'exemple suivant illustre l'utilisation de l'opérateur `-=` avec des variables et des nombres :

```
x -= y is the same as x = x - y
If x = 5 and y = 10 then
x -= 10 returns -5
```

== (égalité)

Syntaxe

expression1 == *expression2*

Arguments

expression1, expression2 Nombres, chaînes, booléens, variables, objets, tableaux ou fonctions.

Description

Opérateur (égalité) ; teste l'égalité de deux expressions. Le résultat est `true` si les deux expressions sont égales.

La définition d'*égales* dépend du type de données de l'argument :

- Les nombres, les chaînes et les valeurs booléennes sont comparés par valeurs et sont considérés identiques s'ils possèdent les mêmes valeurs. Par exemple, deux chaînes sont égales si elles possèdent le même nombre de caractères.
- Les variables, les objets, les tableaux et les fonctions sont comparés par référence. Deux variables sont identiques si elles référent au même objet ou tableau ou à la même fonction. Deux tableaux distincts ne sont jamais considérés égaux, même s'ils comportent le même nombre d'éléments.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise l'opérateur == avec une instruction `if` :

```
a = "David" , b = "David";
if (a == b)
trace("David is David");
```

> (supérieur à)

Syntaxe

expression1 > *expression2*

Arguments

expression1, expression2 Entiers, nombres à virgule flottante ou chaînes.

Description

Opérateur (comparaison) ; compare deux expressions et détermine si *expression1* est supérieure à *expression2* (`true`) ou si *expression1* est inférieure ou égale à *expression2* (`false`).

Lecteur
Flash 4 ; Flash 5 et versions suivantes. Dans Flash 5, > est un opérateur de comparaison capable de manipuler divers types de données. Dans Flash 4, > est un opérateur numérique. Les fichiers Flash 4 importés dans l'environnement de programmation de Flash 5 subissent un processus de conversion pour préserver l'intégrité des types de données. L'exemple ci-dessous illustre le processus de conversion.

Exemple
L'exemple suivant illustre la conversion d'un fichier Flash 4 contenant une comparaison d'égalité numérique.

Fichier Flash 4 :

```
x > y
```

Fichier Flash 5 converti :

```
Number(x) > Number(y)
```

>= (supérieur ou égal à)

Syntaxe
expression1 >= *expression2*

Arguments
expression1, expression2 Chaînes, entiers ou nombres à virgule flottante.

Description
Opérateur (comparaison) ; compare deux expressions et détermine si *expression1* est supérieure ou égale à *expression2* (`true`) ou si *expression1* est inférieure à *expression2* (`false`).

Lecteur
Flash 4 ; Flash 5 et versions suivantes. Dans Flash 5, >= est un opérateur de comparaison capable de manipuler divers types de données. Dans Flash 4, >= est un opérateur numérique. Les fichiers Flash 4 importés dans l'environnement de programmation de Flash 5 subissent un processus de conversion pour préserver l'intégrité des types de données. L'exemple ci-dessous illustre le processus de conversion.

Exemple
L'exemple suivant illustre la conversion d'un fichier Flash 4 contenant une comparaison d'égalité numérique.

Fichier Flash 4 :

```
x >= y
```

Fichier Flash 5 converti :

```
Number(x) >= Number(y)
```

>> (décalage droit au niveau du bit)

Syntaxe

expression1 >> *expression2*

Arguments

expression1 Un nombre, une chaîne ou une expression devant être décalés à droite.

expression2 Un nombre, une chaîne ou une expression convertis en entier de 0 à 31.

Description

Opérateur (au niveau du bit) ; convertit *expression1* et *expression2* en entiers de 32 bits et décale tous les bits de *expression1* vers la droite du nombre de places spécifié par l'entier résultant de la conversion de *expression2*. Les bits qui sont décalés vers la droite sont éliminés. Pour préserver le signe de l'*expression* d'origine, les bits de gauche sont remplis de 0 si le bit le plus significatif (le bit le plus à gauche) de *expression1* est 0 et remplis de 1 si le bit le plus significatif est 1. Décaler une valeur à droite de 1 revient à la diviser par 2 et à éliminer le reste.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant convertit 65535 en entier de 32 bits et le décale vers la droite de huit bits.

```
x = 65535 >> 8
```

Le résultat de cette opération est :

```
x = 255
```

Ceci est dû au fait que 65535 décimal égale 1111111111111111 binaire (*seize 1*), 1111111111111111 binaire décalé à droite de huit bits est 11111111 binaire, et 11111111 binaire est 255 décimal. Le bit le plus significatif est 0 car les entiers ont 32 bits, donc le bit de remplissage est 0.

L'exemple suivant convertit -1 en entier de 32 bits et le décale vers la droite de un bit.

```
x = -1 >> 1
```

Le résultat de cette opération est :

```
x = -1
```

Ceci est dû au fait que -1 décimal égale 11111111111111111111111111111111 binaire (trente-deux 1), le décalage vers la droite de un bit fait que le bit le moins significatif (le bit le plus à droite) est éliminé et que le bit le plus significatif est rempli de 1. Le résultat est 11111111111111111111111111111111 (*trente-deux 1*) binaire, ce qui représente l'entier de 32 bits -1.

Voir aussi

« >>= (décalage droit au niveau du bit et affectation) » à la page 219

>>= (décalage droit au niveau du bit et affectation)

Syntaxe

expression1 =>>*expression2*

Arguments

expression1 Un nombre, une chaîne ou une expression devant être décalés à droite.

expression2 Un nombre, une chaîne ou une expression convertis en entier de 0 à 31.

Description

Opérateur (affectation composée) ; cet opérateur effectue une opération de décalage vers la droite au niveau du bit et stocke le contenu comme résultat dans *expression1*.

Lecteur

Flash 5 et versions suivantes.

Exemple

Les deux expressions suivantes sont identiques.

```
A >>= B
A = (A >> B)
```

Le code commenté suivant utilise l'opérateur au niveau du bit >>=. Cet exemple illustre également l'utilisation de tous les opérateurs au niveau du bit.

```
function convertToBinary(number)
{
var result = "";
for (var i=0; i<32; i++) {
// Extract least significant bit using bitwise AND
var lsb = number & 1;
// Add this bit to our result string
result = (lsb ? "1" : "0") + result;
// Shift number right by one bit, to see next bit
}number >>= 1;
return result;
}
convertToBinary(479)
//Returns the string
00000000000000000000000111011111
//The above string is the binary representation of the decimal
number 479.
```

Voir aussi

« << (décalage gauche au niveau du bit) » à la page 211

>>> (décalage droit non signé au niveau du bit)

Syntaxe

expression1 >>> *expression2*

Arguments

expression1 Un nombre, une chaîne ou une expression devant être décalés vers la droite.

expression2 Un nombre, une chaîne ou une expression convertis en entier de 0 à 31.

Description

Opérateur (au niveau du bit) ; identique à l'opérateur de décalage droit au niveau du bit (>>) excepté qu'il ne conserve pas le signe de l'*expression* d'origine car les bits de gauche sont toujours remplis avec des 0.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant convertit -1 en un entier de 32 bits et le décale vers la droite de un bit.

```
x = -1 >>> 1
```

Le résultat de l'opération précédente est :

```
x = 2147483647
```

Cela est dû au fait que -1 décimal est 11111111111111111111111111111111 binaire (*trente-deux 1*) et, quand vous le décalez vers la droite (non signé) de un bit, le bit le moins significatif (le plus à droite) est éliminé et le bit le plus significatif (le plus à gauche) est rempli avec un 0. Le résultat est :

01111111111111111111111111111111 binaire,

qui représente l'entier de 32 bits `2147483647`.

Voir aussi

« >>= (décalage droit au niveau du bit et affectation) » à la page 219

>>>= (décalage droit non signé au niveau du bit et affectation)

Syntaxe

expression1 >>>= *expression2*

Arguments

expression1 Un nombre, une chaîne ou une expression devant être décalés vers la droite.

expression2 Un nombre, une chaîne ou une expression convertis en entier de 0 à 31.

Description

Opérateur (affectation composée) ; effectue une opération de décalage vers la droite non signé et stocke le contenu comme résultat dans *expression1*.

Lecteur

Flash 5 et versions suivantes.

Exemple

Les deux expressions suivantes sont identiques.

```
A >>>= B
A = (A >>> B)
```

Voir aussi

« >>> (décalage droit non signé au niveau du bit) » à la page 220
« >>= (décalage droit au niveau du bit et affectation) » à la page 219

add

Syntaxe

```
string1 add string2
```

Arguments

string1, string2 Toute chaîne.

Description

Opérateur ; concatène deux ou plusieurs chaînes. L'opérateur `add` remplace l'opérateur `&` de Flash 4 ; les fichiers de Flash 4 utilisant l'opérateur `&` sont automatiquement convertis pour utiliser l'opérateur `add` pour concaténer les chaînes, lorsqu'ils sont importés vers l'environnement de programmation de Flash 5. Cependant, l'opérateur `add` est désapprouvé dans Flash 5 et l'usage de l'opérateur + est recommandé lors de la création d'un contenu pour le Flash 5 Player. Utilisez l'opérateur `add` pour concaténer les chaînes si vous créez un contenu pour Flash 4 ou pour les premières versions du Player.

Lecteur

Flash 4 et versions suivantes.

Voir aussi

« + (addition) » à la page 209

_alpha

Syntaxe

```
instancename._alpha
instancename._alpha = value;
```

Arguments

instancename Le nom d'une occurrence de clip.

value Un nombre entre 0 et 100 spécifiant la transparence alpha.

Description

Propriété ; définit ou extrait la transparence alpha (*value*) du clip. Les valeurs correctes sont comprises entre 0 (complètement transparent) et 100 (complètement opaque). Les objets d'un clip dont `_alpha` est définie sur 0 sont actifs, même s'ils sont invisibles. Par exemple, un bouton dans un clip dont la propriété `_alpha` est définie sur 0 peut toujours être activé.

Lecteur

Flash 4 et versions suivantes.

Exemple

Les instructions suivantes définissent la propriété `_alpha` d'un clip nommé `star` sur 30% lorsque le bouton est enfoncé.

```
on(release) {
   setProperty(star._alpha = 30);
}
```

ou

```
on(release) {
   star._alpha = 30;
}
```

and

Syntaxe

condition1 and *condition2*

Arguments

condition1, condition2 Conditions ou expressions qui évaluent sur `true` ou `false`.

Description

Opérateur ; effectue une opération logique AND dans le Flash 4 Player. Si les deux expressions évaluent sur `true`, alors l'expression entière est `true`.

Lecteur

Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 et les utilisateurs sont encouragés à utiliser le nouvel opérateur `&&`.

Voir aussi

« && (AND court-circuit) » à la page 194

Tableau (objet)

L'objet Tableau vous permet d'accéder à des tableaux et de les manipuler. Un tableau est un objet dont les propriétés sont identifiées par un nombre représentant leur position dans le tableau. Ce nombre est parfois appelé index. Tous les tableaux sont basés sur zéro, ce qui signifie que le premier élément du tableau est [0], le second élément est [1], et ainsi de suite. Dans l'exemple suivant, `myArray` contient les mois de l'année, identifiés par des nombres.

```
myArray[0] = "January"
myArray[1] = "February"
myArray[2] = "March"
myArray[3] = "April"
```

Pour créer un objet Tableau, utilisez le constructeur `new Array`. Pour accéder aux éléments d'un tableau, utilisez l'opérateur accès tableau [ ].

Tableau des méthodes de l'objet Tableau

Méthode	Description
`concat`	Concatène les arguments et les renvoie sous forme de nouveau tableau.
`join`	Joint tous les éléments d'un tableau dans une chaîne.
`pop`	Supprime le dernier élément d'un tableau et renvoie sa valeur.
`push`	Ajoute un ou plusieurs éléments à la fin d'un tableau et renvoie la nouvelle longueur du tableau.
`reverse`	Inverse le sens d'un tableau.
`shift`	Supprime le premier élément d'un tableau et renvoie sa valeur.
`slice`	Extrait une section d'un tableau et la renvoie sous forme de nouveau tableau.
`sort`	Trie un tableau en place.
`splice`	Ajoute et/ou supprime des éléments d'un tableau.
`toString`	Renvoie une valeur de chaîne représentant les éléments dans l'objet Tableau.
`unshift`	Ajoute un ou plusieurs éléments au début d'un tableau et renvoie la nouvelle longueur du tableau.

Tableau des propriétés de l'objet Tableau

Propriété	Description
`length`	Renvoie la longueur du tableau.

Constructeur de l'objet Tableau

Syntaxe

```
new Array();

new Array(length);

new Array(element0, element1, element2,...elementN);
```

Arguments

length Un entier spécifiant le nombre d'éléments dans un tableau. Dans le cas d'éléments discontinus, la longueur spécifie le numéro d'index du dernier élément du tableau plus 1. Pour plus d'informations, voir la propriété `Array.length`.

element0...elementN Une liste de deux ou plusieurs valeurs arbitraires. Les valeurs peuvent être des nombres, des noms ou tout autre élément spécifié dans un tableau. Le premier élément d'un tableau possède toujours l'index ou la position 0.

Description

Constructeur ; vous permet d'accéder aux éléments d'un tableau et de les manipuler. Les tableaux sont basés sur zéro et les éléments sont indexés selon leur nombre ordinal.

Si vous ne spécifiez pas d'arguments, un tableau de longueur zéro sera créé.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant crée un nouvel objet Tableau avec une longueur initiale 0.

```
myArray = new Array();
```

L'exemple suivant crée le nouvel objet Chaîne `A-Team`, avec une longueur initiale de 4.

```
A-Team = new Array("Jody", "Mary", "Marcelle", "Judy");
```

Les éléments initiaux du tableau `A-Team` sont :

```
myArray[0] = "Jody"
myArray[1] = "Mary"
myArray[2] = "Marcelle"
myArray[3] = "Judy"
```

Voir aussi

« Array.length » à la page 227

Array.concat

Syntaxe

myArray.concat(*value0,value1,...valueN*);

Arguments

value0,...valueN Nombres, éléments ou chaînes devant être concaténés dans un nouveau tableau.

Description

Méthode ; concatène les éléments spécifiés dans les arguments, s'il y en a, et crée et renvoie un nouveau tableau. Si les arguments spécifient un tableau, ce sont les éléments de ce tableau qui sont concaténés et non le tableau lui-même.

Lecteur

Flash 5 et versions suivantes.

Exemple

Le code suivant concatène deux tableaux :

```
alpha = new Array("a","b","c");
numeric = new Array(1,2,3);
alphaNumeric=alpha.concat(numeric); // creates array
["a","b","c",1,2,3]
```

Le code suivant concatène trois tableaux :

```
num1=[1,3,5];
num2=[2,4,6];
num3=[7,8,9];
nums=num1.concat(num2,num3) // creates array [1,3,5,2,4,6,7,8,9]
```

Array.join

Syntaxe

myArray.join();

myArray.join(*separator*);

Arguments

separator Un caractère ou une chaîne qui sépare les éléments du tableau dans la chaîne renvoyée. Si vous oubliez cet argument, une virgule est utilisée comme séparateur par défaut.

Description

Méthode ; convertit les éléments d'un tableau en chaînes, les concatène, insère le séparateur spécifié entre les éléments et renvoie la chaîne résultante.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant crée un tableau avec trois éléments. Il joint ensuite le tableau trois fois : en utilisant le séparateur par défaut, puis avec une virgule suivie d'un espace et, pour finir, avec un signe plus.

```
a = new Array("Earth","Moon","Sun")
// assigns "Earth,Moon,Sun" to myVar1
myVar1=a.join();
// assigns "Earth, Moon, Sun" to myVar2
myVar2=a.join(", ");
// assigns "Earth + Moon + Sun" to myVar3
myVar3=a.join(" + ");
```

Array.length

Syntaxe
myArray.length;

Arguments
Aucun.

Description
Propriété ; contient la longueur du tableau. Cette propriété est automatiquement mise à jour lorsque de nouveaux éléments sont ajoutés dans le tableau. Lors d'une affectation `myArray[index] = value`, si `index` est un nombre et `index+1` est supérieur à la propriété `length`, la propriété `length` est mise à jour avec `index + 1`.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant explique comment la propriété length est mise à jour.

```
//initial length is 0
myArray = new Array();
//myArray.length is updated to 1
myArray[0] = 'a'
//myArray.length is updated to 2
myArray[1] = 'b'
//myArray.length is updated to 10
myArray[9] = 'c'
```

Array.pop

Syntaxe
myArray.pop();

Arguments
Aucun.

Description
Méthode ; supprime le dernier élément d'un tableau et renvoie la valeur de cet élément.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant crée le tableau `myPets` contenant quatre éléments et supprime le dernier élément.

```
myPets = ["cat", "dog", "bird", "fish"];
popped = myPets.pop();
```

Array.push

Syntaxe
myArray.push(*value,...*);

Arguments
value Une ou plusieurs valeurs à ajouter à la fin du tableau.

Description
Méthode ; ajoute un ou plusieurs éléments à la fin du tableau et renvoie la nouvelle longueur du tableau.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant crée le tableau `myPets` contenant deux éléments, puis lui ajoute deux éléments. Après l'exécution du code, `pushed` contient 4.

```
myPets = ["cat", "dog"];
pushed = myPets.push("bird", "fish")
```

Array.reverse

Syntaxe
myArray.reverse();

Arguments
Aucun.

Description
Méthode ; inverse le tableau en place.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant utilise la méthode `Array.reverse`.

```
var numbers = [1, 2, 3, 4, 5, 6];
trace(numbers.join())
 numbers.reverse()
 trace(numbers.join())
```

Résultat :

```
1,2,3,4,5,6
6,5,4,3,2,1
```

Array.shift

Syntaxe
myArray.shift();

Arguments
Aucun.

Description
Méthode ; supprime le premier élément d'un tableau et renvoie sa valeur.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant crée le tableau `myPets` puis supprime le premier élément du tableau :

```
myPets = ["cat", "dog", "bird", "fish"];
shifted = myPets.shift();
```

La valeur renvoyée est `cat`.

Voir aussi
« Array.pop » à la page 228

« Array.unshift » à la page 233

Array.slice

Syntaxe

```
myArray.slice(start, end);
```

Arguments

start Un nombre spécifiant le point de départ de la section. Si *start* est un nombre négatif, le point de départ commence à la fin du tableau, où -1 est le dernier élément.

end Un nombre spécifiant le point d'arrivée de la section. Si vous oubliez cet argument, la section comprend tous les éléments du début à la fin du tableau. Si *end* est un nombre négatif, le point d'arrivée est spécifié depuis la fin du tableau, où –1 est le dernier élément.

Description

Méthode ; extrait une section ou une sous-chaîne du tableau et la renvoie sous forme de nouveau tableau sans modifier le tableau original. Le tableau renvoyé comprend l'élément *start* et tous les éléments de la section excepté l'élément *end*.

Lecteur

Flash 5 et versions suivantes.

Array.sort

Syntaxe

```
myArray.sort();
myArray.sort(orderfunc);
```

Arguments

orderfunc Une fonction de comparaison optionnelle utilisée pour déterminer l'ordre de tri. Étant donnés les arguments A et B, la fonction de tri spécifiée effectuera un tri selon les critères suivants :

- -1 si A apparaît avant B dans la séquence triée
- 0 si A = B
- 1 si A apparaît après B dans la séquence triée

Description

Méthode ; trie le tableau en place sans en faire de copie. Si vous oubliez l'argument *orderfunc*, Flash trie les éléments en place en utilisant l'opérateur de comparaison <.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `Array.sort` sans spécifier l'argument *`orderfunc`*.

```
var fruits = ["oranges", "apples", "strawberries",
                "pineapples", "cherries"];
 trace(fruits.join())
 fruits.sort()
 trace(fruits.join())
```

Résultat :

```
 oranges,apples,strawberries,pineapples,cherries
 apples,cherries,oranges,pineapples,strawberries
```

L'exemple suivant utilise `array.sort` avec une fonction de tri spécifiée.

```
var passwords = [
   "gary:foo",
   "mike:bar",
   "john:snafu",
   "steve:yuck",
   "daniel:1234"
 ];
 function order (a, b) {
   // Entries to be sorted are in form
   // name:password
   // Sort using only the name part of the
   // entry as a key.
   var name1 = a.split(':')[0];
   var name2 = b.split(':')[0];
   if (name1 < name2) {
     return -1;
   } else if (name1 > name2) {
     return 1;
   } else {
     return 0;
   }
 }
 for (var i=0; i< password.length; i++) {
   trace (passwords.join());
 }
 passwords.sort(order);
 trace ("Sorted:")
 for (var i=0; i< password.length; i++) {
   trace (passwords.join());
 }
```

Résultat :

```
 daniel:1234
 gary:foo
 john:snafu
 mike:bar
 steve:yuck
```

Array.splice

Syntaxe

```
myArray.splice(start, deleteCount, value0,value1...valueN);
```

Arguments

start L'index de l'élément du tableau à partir duquel commencent l'insertion et/ou la suppression.

deleteCount Le nombre d'éléments à supprimer. Ce nombre comprend l'élément spécifié dans l'argument *start*. Si aucune valeur n'est spécifiée pour *deleteCount*, la méthode supprime toutes les valeurs en commençant par l'élément *start* jusqu'au dernier élément du tableau.

value Zéro ou plusieurs valeurs à insérer dans le tableau au point d'insertion spécifié dans l'argument *start*. Cet argument est optionnel.

Description

Méthode ; ajoute et/ou supprime les éléments d'un tableau. Cette méthode modifie le tableau lui-même sans en faire de copie.

Lecteur

Flash 5 et versions suivantes.

Array.toString

Syntaxe

```
myArray.toString();
```

Arguments

Aucun.

Description

Méthode ; renvoie une valeur de chaîne représentant les éléments dans l'objet Tableau spécifié. Chaque élément du tableau, commençant par l'index 0 et terminant par l'index `myArray.length-1`, est converti en chaîne concaténée séparée par des virgules.

Lecteur

Flash 5 et versions suivantes.

Exemple
L'exemple suivant crée `myArray` et le convertit en chaîne.

```
myArray = new Array();
myArray[0] = 1;
myArray[1] = 2;
myArray[2] = 3;
myArray[3] = 4;
myArray[4] = 5;

trace(myArray.toString())
```

Résultat :

```
1,2,3,4,5
```

Array.unshift

Syntaxe
myArray.unshift(*value1,value2,...valueN*);

Arguments
value1,...valueN Un ou plusieurs nombres, éléments ou variables devant être insérés au début du tableau.

Description
Méthode ; ajoute un ou plusieurs éléments au début d'un tableau et renvoie la nouvelle longueur du tableau.

Lecteur
Flash 5 et versions suivantes.

Boolean (fonction)

Syntaxe
Boolean(*expression*);

Arguments
expression La variable, le nombre ou la chaîne devant être convertis en booléen.

Description
Fonction ; convertit l'argument spécifié en booléen et renvoie la valeur booléenne.

Lecteur
Flash 5 et versions suivantes.

Booléen (objet)

L'objet Booléen est un simple objet enveloppe ayant la même fonctionnalité que l'objet Booléen standard de JavaScript. Utilisez l'objet Booléen pour extraire le type de données primitif ou la représentation chaîne d'objets Booléen.

Tableau des méthodes de l'objet Boolén

Méthode	Description
`toString`	Renvoie la représentation chaîne `(true)` ou `(false)` de l'objet Booléen.
`valueOf`	Renvoie le type de valeur primitif de l'objet Booléen spécifié.

Constructeur de l'objet Booléen

Syntaxe

```
new Boolean();
new Boolean(x);
```

Arguments

x Un nombre, une chaîne, un booléen, un objet, un clip ou une autre expression. Cet argument est optionnel.

Description

Constructeur ; crée une occurrence de l'objet Booléen. Si vous oubliez l'argument *x*, l'objet Booléen est initialisé avec la valeur `false`. Si vous spécifiez *x*, la méthode évalue l'argument et renvoie le résultat sous forme de valeur booléenne selon les règles suivantes.

- Si *x* est un nombre, la fonction renvoie `true` si *x* n'est pas égal à 0, ou `false` si *x* est un autre nombre.
- Si *x* est un booléen, la fonction renvoie *x*.
- Si *x* est un objet ou un clip, la fonction renvoie `true` si *x* n'est pas égal à `null` ; sinon, la fonction renvoie `false`.
- Si *x* est une chaîne, la fonction renvoie `true` si `Number(x)` n'est pas égal à 0 ; sinon, la fonction renvoie `false`.

Remarque : Pour préserver la compatibilité avec Flash 4, la manipulation des chaînes avec l'objet Booléen n'est pas à la norme ECMA-262.

Lecteur

Flash 5 et versions suivantes.

Boolean.toString

Syntaxe
```
Boolean.toString();
```

Arguments
Aucun.

Description
Méthode ; renvoie la représentation chaîne,`true` ou `false`, de l'objet Booléen.

Lecteur
Flash 5 et versions suivantes.

Boolean.valueOf

Syntaxe
```
Boolean.valueOf();
```

Arguments
Aucun.

Description
Méthode ; renvoie le type de valeur primitif de l'objet Booléen spécifié et convertit l'objet enveloppe Booléen en ce type de valeur primitif.

Lecteur
Flash 5 et versions suivantes.

break

Syntaxe
```
break;
```

Arguments
Aucun.

Description
Action ; apparaît à l'intérieur d'une boucle (`for`, `for..in`, `do...while` ou `while`). L'action `break` indique à Flash de passer le reste du corps de la boucle, de stopper l'action de boucle et d'exécuter l'instruction qui suit l'instruction de boucle. Utilisez l'action `break` pour sortir d'une série de boucles imbriquées.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant utilise l'action `break` pour sortir d'une boucle sans fin.

```
i = 0;
while (true) {
  if (i >= 100) {
    break;
  }
  i++;
}
```

call

Syntaxe
`call(`*`frame`*`);`

Arguments
`frame` Le numéro ou le nom de l'image à appeler dans le contexte du script.

Description
Action ; bascule le contexte du script en cours vers le script associé à l'image appelée. Les variables locales ne sortiront pas une fois l'exécution du script terminée.

Lecteur
Flash 4 et versions suivantes. Cette action est désapprouvée dans Flash 5 et il est recommandé d'utiliser l'action `function`.

Voir aussi
« function » à la page 271

chr

Syntaxe
`chr(`*`number`*`);`

Arguments
`number` Le nombre du code ASCII à convertir en caractère.

Description
Fonction de chaîne ; convertit les nombres des codes ASCII en caractères.

Lecteur
Flash 4 et versions suivantes. Cette fonction est désapprouvée dans Flash 5 ; l'utilisation de la méthode `String.fromCharCode` est recommandée.

Exemple
L'exemple suivant convertit le nombre 65 en lettre « A ».

```
chr(65) = "A"
```

Voir aussi
« String.fromCharCode » à la page 374

Couleur (objet)

L'objet Couleur vous permet de définir et d'extraire la valeur RGB et la transformation d'une couleur dans un clip. L'objet Couleur est supporté par Flash 5 et les versions suivantes du Flash Player.

Vous devez utiliser le constructeur `new Color()` pour créer une occurrence de l'objet Couleur avant d'appeler les méthodes de l'objet Couleur.

Tableau des méthodes de l'objet Couleur

Méthode	Description
`getRGB`	Renvoie la valeur numérique RGB définie par le dernier appel `setRGB`.
`getTransform`	Renvoie les informations de transformation définies par le dernier appel `setTransform`.
`setRGB`	Définit la représentation hexadécimale de la valeur RGB pour un objet Couleur.
`setTransform`	Définit la transformation de couleur pour un objet Couleur.

Constructeur de l'objet Couleur

Syntaxe
`new Color(`*`target`*`);`

Arguments
`target` Le nom du clip auquel est appliquée la nouvelle couleur.

Description
Constructeur ; crée un objet Couleur pour le clip spécifié par l'argument *`target`*.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant crée un nouvel objet Couleur appelé `myColor` pour l'animation `myMovie`.

```
myColor = new Color(myMovie);
```

Color.getRGB

Syntaxe
```
myColor.getRGB();
```

Arguments
Aucun.

Description
Méthode ; renvoie les valeurs numériques définies par le dernier appel `setRGB`.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant extrait la valeur RGB sous forme d'une chaîne hexadécimale :

```
value = (getRGB()).toString(16);
```

Voir aussi
« Color.setRGB » à la page 239

Color.getTransform

Syntaxe
```
myColor.getTransform();
```

Arguments
Aucun.

Description
Méthode ; renvoie la valeur de transformation définie par le dernier appel `setTransform`.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« Color.setTransform » à la page 239

Color.setRGB

Syntaxe

myColor.setRGB(*0xRRGGBB*);

Arguments

0xRRGGBB La couleur hexadécimale ou RGB à définir. *RR*, *GG* et *BB* sont constitués, chacun, de deux chiffres hexadécimaux spécifiant le décalage de chaque composant de couleur.

Description

Méthode ; spécifie une couleur RGB pour un objet Couleur. L'appel de cette méthode a la priorité sur toutes les définitions de la méthode `setTransform`.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant définit la valeur de la couleur RGB pour le clip `myMovie`.

```
myColor = new Color(myMovie);
myColor.setRGB(0x993366);
```

Voir aussi

« Color.setTransform » à la page 239

Color.setTransform

Syntaxe

myColor.setTransform(*colorTransformObject*);

Arguments

colorTransformObject Un objet créée avec le constructeur de l'objet générique Objet, spécifiant les valeurs de transformation de couleur comme paramètres. L'objet colorTransform doit avoir les paramètres `ra`, `rb`, `ga`, `gb`, `ba`, `bb`, `aa`, `ab`, qui sont expliqués ci-dessous.

Description

Méthode ; définit les informations de transformation de couleur pour un objet Couleur. L'argument *colorTransformObject* est un objet que vous créez en utilisant l'objet générique Objet avec des paramètres spécifiant les valeurs de pourcentage et de décalage pour les composants rouge, vert, bleu et alpha (transparence) d'une couleur, entrés sous le format *0xRRGGBBAA*.

Les paramètres d'un objet colorTransform sont définis selon les règles suivantes :

- *ra* est le pourcentage du composant rouge (-100 à 100).
- *rb* est le décalage du composant rouge (-255 à 255).
- *ga* est le pourcentage du composant vert (-100 à 100).
- *gb* est le décalage du composant vert (-255 à 255).
- *ba* est le pourcentage du composant bleu (-100 à 100).
- *bb* est le décalage du composant bleu (-255 à 255).
- *aa* est le pourcentage pour alpha (-100 à 100).
- *ab* est le décalage pour alpha (-255 à 255).

Vous créez un objet colorTransform selon l'exemple suivant :

```
myColorTransform = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

Vous pouvez également utiliser la syntaxe suivante :

```
myColorTransform = { ra: '50', rb: '244', ga: '40', gb: '112',
ba: '12', bb: '90', aa: '40', ab: '70'}
```

Lecteur

Flash 5 et versions suivantes.

Exemple
L'exemple suivant montre le processus de création d'un objet nouvelle Couleur pour une animation ciblée, en créant un objet colorTransform avec les paramètres définis ci-dessus en utilisant le constructeur Object et en transmettant l'objet colorTransform à un objet Couleur avec la méthode `setTransform`.

```
//Create a color object called myColor for the target myMovie

myColor = new Color(myMovie);

//Create a color transform object called myColorTransfrom using
the generic object Object

myColorTransform = new Object;

// Set the values for myColorTransform

myColorTransform = { ra: '50', rb: '244', ga: '40', gb: '112', ba:
'12', bb: '90', aa: '40', ab: '70'}

//Associate the color transform object with the Color object
created for myMovie

myColor.setTransform(myColorTransform);
```

continue

Syntaxe
`continue;`

Arguments
Aucun.

Description
Action ; apparaît dans plusieurs types d'instructions de boucle.

Dans une boucle `while`, `continue` oblige Flash à passer le reste du corps de la boucle et à aller au début de la boucle où la condition est testée.

Dans une boucle `do...while`, `continue` oblige Flash à passer le reste du corps de la boucle et à aller en bas de la boucle où la condition est testée.

Dans une boucle `for`, `continue` oblige Flash à passer le reste du corps de la boucle et à aller à l'évaluation de la post-expression de la boucle `for`.

Dans une boucle `for...in`, `continue` oblige Flash à passer le reste du corps de la boucle et à retourner au début de la boucle où la valeur suivante dans l'énumération est traitée.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« do...while » à la page 261
« for » à la page 267
« for..in » à la page 268
« while » à la page 388

_currentframe

Syntaxe
instancename._currentframe

Arguments
instancename Le nom d'une occurrence de clip.

Description
Propriété (lecture seule) ; renvoie le numéro de l'image où se trouve actuellement la tête de lecture dans le scénario.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant utilise _currentframe pour diriger une animation et la faire aller cinq images plus loin que celle où se trouve l'action :

```
gotoAndStop(_currentframe + 5);
```

Date (objet)

L'objet Date vous permet d'extraire les valeurs de date et d'heure relatives au temps universel (Temps Moyen de Greenwich, maintenant appelé Temps Universel Coordonné) ou relatives au système d'exploitation sous lequel le Flash Player est exécuté. Pour appeler les méthodes de l'objet Date, vous devez d'abord créer une occurrence de l'objet Date en utilisant le constructeur.

L'objet Date nécessite le Flash 5 Player.

Les méthodes de l'objet Date ne sont pas statiques, mais elles ne s'appliquent qu'à l'occurrence de l'objet Date spécifiée lors de l'appel de la méthode.

Tableau des méthodes de l'objet Date

Méthode	Description
`getDate`	Renvoie le jour du mois de l'objet Date spécifié en fonction de l'heure locale.
`getDay`	Renvoie le jour du mois pour l'objet Date spécifié en fonction de l'heure locale.
`getFullYear`	Renvoie l'année complète (quatre chiffres) de l'objet Date spécifié en fonction de l'heure locale.
`getHours`	Renvoie l'heure de l'objet Date spécifié en fonction de l'heure locale.
`getMilliseconds`	Renvoie les millisecondes de l'objet Date spécifié en fonction de l'heure locale.
`getMinutes`	Renvoie les minutes de l'objet Date spécifié en fonction de l'heure locale.
`getMonth`	Renvoie le mois de l'objet Date spécifié en fonction de l'heure locale.
`getSeconds`	Renvoie les secondes de l'objet Date spécifié en fonction de l'heure locale.
`getTime`	Renvoie le nombre de millisecondes écoulés depuis le premier janvier 1970 à minuit, temps universel, pour l'objet Date spécifié.
`getTimezoneOffset`	Renvoie la différence, en minutes, entre l'heure locale de l'ordinateur et le temps universel.
`getUTCDate`	Renvoie le jour (date) du mois de l'objet Date spécifié en fonction du temps universel.
`getUTCDay`	Renvoie le jour de la semaine de l'objet Date spécifié en fonction du temps universel.
`getUTCFullYear`	Renvoie l'année complète (quatre chiffres) de l'objet Date spécifié en fonction du temps universel.
`getUTCHours`	Renvoie l'heure de l'objet Date spécifié en fonction du temps universel.
`getUTCMilliseconds`	Renvoie les millisecondes de l'objet Date spécifié en fonction du temps universel.
`getUTCMinutes`	Renvoie les minutes de l'objet Date spécifié en fonction du temps universel.
`getUTCMonth`	Renvoie le mois de l'objet Date spécifié en fonction du temps universel.
`getUTCSeconds`	Renvoie les secondes de l'objet Date spécifié en fonction du temps universel.

Méthode	Description
`getYear`	Renvoie l'année de l'objet Date spécifié en fonction de l'heure locale.
`setDate`	Définit le jour du mois de l'objet Date spécifié en fonction de l'heure locale.
`setFullYear`	Définit l'année complète pour un objet Date en fonction de l'heure locale.
`setHours`	Définit les heures pour un objet Date en fonction de l'heure locale.
`setMilliseconds`	Définit les millisecondes pour un objet Date en fonction de l'heure locale.
`setMinutes`	Définit les minutes pour un objet Date en fonction de l'heure locale.
`setMonth`	Définit le mois pour un objet Date en fonction de l'heure locale.
`setSeconds`	Définit les secondes pour un objet Date en fonction de l'heure locale.
`setTime`	Définit la date pour l'objet Date spécifié en millisecondes.
`setUTCDate`	Définit la date pour l'objet Date spécifié en fonction du temps universel.
`setUTCFullYear`	Définit l'année pour l'objet Date spécifié en fonction du temps universel.
`setUTCHours`	Définit l'heure pour l'objet Date spécifié en fonction du temps universel.
`setUTCMilliseconds`	Définit les millisecondes pour l'objet Date spécifié en fonction du temps universel.
`setUTCMinutes`	Définit la minute pour l'objet Date spécifié en fonction du temps universel.
`setUTCMonth`	Définit le mois représenté par l'objet Date spécifié en fonction du temps universel.
`setUTCSeconds`	Définit les secondes de l'objet Date spécifié en fonction du temps universel.
`setYear`	Définit l'année pour l'objet Date spécifié en fonction de l'heure locale.
`toString`	Renvoie une valeur de chaîne représentant la date et l'heure stockées dans l'objet Date spécifié.
`Date.UTC`	Renvoie le nombre de millisecondes écoulés entre le premier janvier 1970 à minuit, temps universel, et la période spécifiée.

Constructeur de l'objet Date

Syntaxe

```
new Date();
```

```
new Date(year [, month [, date [, hour [, minute [, second [,
millisecond ]]]]]] );
```

Arguments

year Une valeur entre 0 et 99 indique une année entre 1900 et 1999, sinon, les quatre chiffres de l'année doivent être spécifiés.

month Un entier entre 0 (janvier) et 11 (décembre). Cet argument est optionnel.

date Un entier entre 1 et 31. Cet argument est optionnel.

hour Un entier entre 0 (minuit) et 23 (23h00).

minute Un entier entre 0 et 59. Cet argument est optionnel.

second Un entier entre 0 et 59. Cet argument est optionnel.

millisecond Un entier entre 0 et 999. Cet argument est optionnel.

Description

Objet ; construit un nouvel objet Date contenant la date et l'heure en cours.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant extrait l'heure et la date en cours.

```
now = new Date();
```

L'exemple suivant crée un nouvel objet Date pour l'anniversaire de Gary, le 7 août 1974.

```
gary_birthday = new Date (74, 7, 7);
```

L'exemple suivant crée un nouvel objet Date, concatène les valeurs renvoyées des méthodes `getMonth`, `getDate` et `getFullYear` de l'objet Date et les affiche dans le champ de texte spécifié par la variable `dateTextField`.

```
myDate = new Date();
dateTextField = (mydate.getMonth() + "/" + myDate.getDate() + "/"
+ mydate.getFullYear());
```

Date.getDate

Syntaxe

```
myDate.getDate();
```

Arguments

Aucun.

Description
Méthode ; renvoie le jour du mois (un entier entre 1 et 31) de l'objet Date spécifié en fonction de l'heure locale.

Lecteur
Flash 5 et versions suivantes.

Date.getDay

Syntaxe
```
myDate.getDay();
```

Arguments
Aucun.

Description
Méthode ; renvoie le jour du mois (0 pour dimanche, 1 pour lundi, et ainsi de suite) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.getFullYear

Syntaxe
```
myDate.getFullYear();
```

Arguments
Aucun.

Description
Méthode ; renvoie l'année complète (un nombre à quatre chiffres, par exemple, 2000) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant utilise le constructeur pour créer un nouvel objet Date et envoie la valeur renvoyée par la méthode `getFullYear` dans la fenêtre Résultat.

```
myDate = new Date();
trace(myDate.getFullYear());
```

Date.getHours

Syntaxe
```
myDate.getHours();
```

Arguments
Aucun.

Description
Méthode ; renvoie l'heure (un entier entre 0 et 23) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.getMilliseconds

Syntaxe
```
myDate.getMilliseconds();
```

Arguments
Aucun.

Description
Méthode ; renvoie les millisecondes (un entier entre 0 et 999) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.getMinutes

Syntaxe
```
myDate.getMinutes();
```

Arguments
Aucun.

Description
Méthode ; renvoie les minutes (un entier entre 0 et 59) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.getMonth

Syntaxe
```
myDate.getMonth();
```

Arguments
Aucun.

Description
Méthode ; renvoie le mois (0 pour janvier, 1 pour février, et ainsi de suite) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.getSeconds

Syntaxe
```
myDate.getSeconds();
```

Arguments
Aucun.

Description
Méthode ; renvoie les secondes (un entier entre 0 et 59) de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.getTime

Syntaxe
```
myDate.getTime();
```

Arguments
Aucun.

Description
Méthode ; renvoie le nombre de millisecondes (un entier entre 0 et 999) écoulés depuis le premier janvier 1970 à minuit, temps universel, pour l'objet Date spécifié. Utilisez cette méthode pour représenter un instant spécifique du temps lors d'une comparaison entre deux ou plusieurs dates définies dans différentes zones de temps.

Lecteur
Flash 5 et versions suivantes.

Date.getTimezoneOffset

Syntaxe
```
mydate.getTimezoneOffset();
```

Arguments
Aucun.

Description
Méthode ; renvoie la différence, en minutes, entre l'heure locale de l'ordinateur et le temps universel.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant renvoie la différence entre l'heure avancée d'été de San Francisco et le temps universel. L'heure avancée d'été est indicée dans le résultat renvoyé
uniquement si la date définie dans l'objet Date se situe pendant l'heure avancée d'été.

```
new Date().getTimezoneOffset();
```

Le résultat est le suivant :

```
420 (7 hours * 60 minutes/hour = 420 minutes)
```

Date.getUTCDate

Syntaxe
```
myDate.getUTCDate();
```

Arguments
Aucun.

Description
Méthode ; renvoie le jour (date) du mois dans l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getUTCDay

Syntaxe
```
myDate.getUTCDate();
```

Arguments
Aucun.

Description
Méthode ; renvoie le jour de la semaine de l'objet Date spécifié en fonction du temps universel.

Date.getUTCFullYear

Syntaxe
```
myDate.getUTCFullYear();
```

Arguments
Aucun.

Description
Méthode ; renvoie l'année complète (quatre chiffres) de l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getUTCHours

Syntaxe
```
myDate.getUTCHours();
```

Arguments
Aucun.

Description
Méthode ; renvoie les heures de l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getUTCMilliseconds

Syntaxe
```
myDate.getUTCMilliseconds();
```

Arguments
Aucun.

Description
Méthode ; renvoie les millisecondes de l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getUTCMinutes

Syntaxe
```
myDate.getUTCMinutes();
```

Arguments
Aucun.

Description
Méthode ; renvoie les minutes de l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getUTCMonth

Syntaxe
```
myDate.getUTCMonth();
```

Arguments
Aucun.

Description
Méthode ; renvoie le mois de l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getUTCSeconds

Syntaxe
```
myDate.getUTCSeconds();
```

Arguments
Aucun.

Description
Méthode ; renvoie les secondes dans l'objet Date spécifié en fonction du temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.getYear

Syntaxe
```
myDate.getYear();
```

Arguments
Aucun.

Description
Méthode ; renvoie l'année de l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté. L'année est l'année complète moins 1900. Par exemple, l'an 2000 est représenté par 100.

Lecteur
Flash 5 et versions suivantes.

Date.setDate

Syntaxe
```
myDate.setDate(date);
```

Arguments
date Un entier entre 1 et 31.

Description
Méthode ; définit le jour du mois pour l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.setFullYear

Syntaxe
```
myDate.setFullYear(year [, month [, date]] );
```

Arguments
year Un nombre à quatre chiffres spécifiant une année. Les nombres à deux chiffres ne représentent pas une année ; par exemple, 99 n'est pas l'année 1999, mais l'année 99.

month Un entier entre 0 (janvier) et 11 (décembre). Cet argument est optionnel.

date Un nombre entre 1 et 31. Cet argument est optionnel.

Description
Méthode ; définit l'année de l'objet Date spécifié en fonction de l'heure locale. Si les arguments *month* et *date* sont spécifiés, ils sont également définis sur l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Les résultats de `getUTCDay` et de `getDay` peuvent changer suite à l'appel de cette méthode.

Lecteur
Flash 5 et versions suivantes.

Date.setHours

Syntaxe
```
myDate.setHours(hour);
```

Arguments
hour Un entier entre 0 (minuit) et 23 (23h00).

Description
Méthode ; définit les heures pour l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.setMilliseconds

Syntaxe
```
myDate.setMilliseconds(millisecond);
```

Arguments
millisecond Un entier entre 0 et 999.

Description
Méthode ; définit les millisecondes pour l'objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.setMinutes

Syntaxe
```
myDate.setMinutes(minute);
```

Arguments
minute Un entier entre 0 et 59.

Description
Méthode ; définit les minutes pour un objet Date spécifié en fonction de l'heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.setMonth

Syntaxe
```
myDate.setMonth(month [, date ]);
```

Arguments
month Un entier entre 0 (janvier) et 11 (décembre).

date Un entier entre 1 et 31. Cet argument est optionnel.

Description
Méthode ; définit le mois pour l'objet Date spécifié en heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.setSeconds

Syntaxe
```
myDate.setSeconds(second);
```

Arguments
second Un entier entre 0 et 59.

Description
Méthode ; définit les secondes pour l'objet Date spécifié en heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur
Flash 5 et versions suivantes.

Date.setTime

Syntaxe
myDate.setTime(*millisecond*);

Arguments
millisecond Un entier entre 0 et 999.

Description
Méthode ; définit la date pour l'objet Date spécifié en millisecondes.

Lecteur
Flash 5 et versions suivantes.

Date.setUTCDate

Syntaxe
myDate.setUTCDate(*date*);

Arguments
date Un entier entre 1 et 31.

Description
Méthode ; définit la date pour l'objet Date spécifié en temps universel. Appeler cette méthode ne modifie pas les autres champs de la date spécifiée, mais les méthodes `getUTCDay` et `getDay` peuvent rapporter une nouvelle valeur si le jour de la semaine change suite à l'appel de cette méthode.

Lecteur
Flash 5 et versions suivantes.

Date.setUTCFullYear

Syntaxe
myDate.setUTCFullYear(*year* [, *month* [, *date*]]);

Arguments
year L'année spécifiée avec un nombre à quatre chiffres, par exemple, 2000.

month Un entier entre 0 (janvier) et 11 (décembre). Cet argument est optionnel.

date Un entier entre 1 et 31. Cet argument est optionnel.

Description
Méthode ; définit l'année de l'objet Date spécifié (*mydate*) en temps universel.

En option, cette méthode peut également définir le mois et la date représentés par l'objet Date spécifié. Aucun autre champ de l'objet Date n'est modifié. Appeler `setUTCFullyear` peut obliger `getUTCDay` et `getDay` à rapporter une nouvelle valeur si le jour de la semaine change suite à cette opération.

Lecteur
Flash 5 et versions suivantes.

Date.setUTCHours

Syntaxe
```
myDate.setUTCHours(hour [, minute [, second [, millisecond]]]));
```

Arguments
hour Un entier entre 0 (minuit) et 23 (23h00).

minute Un entier entre 0 et 59. Cet argument est optionnel.

second Un entier entre 0 et 59. Cet argument est optionnel.

millisecond Un entier entre 0 et 999. Cet argument est optionnel.

Description
Méthode ; définit l'heure pour l'objet Date spécifié en temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.setUTCMilliseconds

Syntaxe
```
myDate.setUTCMilliseconds(millisecond);
```

Arguments
millisecond Un entier entre 0 et 999.

Description
Méthode ; définit les millisecondes pour l'objet Date spécifié en temps universel.

Lecteur
Flash 5 et versions suivantes.

Date.setUTCMinutes

Syntaxe

```
myDate.setUTCMinutes(minute [, second [, millisecond]]));
```

Arguments

minute Un entier entre 0 et 59.

second Un entier entre 0 et 59. Cet argument est optionnel.

millisecond Un entier entre 0 et 999. Cet argument est optionnel.

Description

Méthode ; définit la minute pour l'objet Date spécifié en temps universel.

Lecteur

Flash 5 et versions suivantes.

Date.setUTCMonth

Syntaxe

```
myDate.setUTCMonth(month [, date]);
```

Arguments

month Un entier entre 0 (janvier) et 11 (décembre).

date Un entier entre 1 et 31. Cet argument est optionnel.

Description

Méthode ; définit le mois, et le jour (date) en option, pour l'objet Date spécifié en temps universel. Appeler cette méthode ne modifie pas les autres champs de l'objet Date spécifié, mais les méthodes `getUTCDay` et `getDay` peuvent rapporter une nouvelle valeur si le jour de la semaine change suite à la spécification de l'argument *date* lors de l'appel de `setUTCMonth`.

Lecteur

Flash 5 et versions suivantes.

Date.setUTCSeconds

Syntaxe

```
myDate.setUTCSeconds(second [, millisecond]));
```

Arguments

second Un entier entre 0 et 59.

millisecond Un entier entre 0 et 999. Cet argument est optionnel.

Description

Méthode ; définit les secondes pour l'objet Date spécifié en temps universel.

Lecteur

Flash 5 et versions suivantes.

Date.setYear

Syntaxe

```
myDate.setYear(year);
```

Arguments

year Un nombre à quatre chiffres, par exemple, 2000.

Description

Méthode ; définit l'année de l'objet Date spécifié en heure locale. L'heure locale est déterminée par le système d'exploitation sous lequel le Flash Player est exécuté.

Lecteur

Flash 5 et versions suivantes.

Date.toString

Syntaxe

```
myDate.toString();
```

Arguments

Aucun.

Description

Méthode ; renvoie une valeur de chaîne pour l'objet Date spécifié dans un format lisible.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant renvoie les informations de l'objet Date dateOfBirth sous forme de chaîne.

```
var dateOfBirth = newDate(74, 7, 7, 18, 15);
trace (dateOfBirth.toString());
```

Résultat (pour le Temps Standard du Pacifique) :

```
Wed Aug 7 18:15:00 GMT-0700 1974
```

Date.UTC

Syntaxe
`Date.UTC(`*`year`*`, `*`month`*` [, `*`date`*` [, `*`hour`*` [, `*`minute`*` [, `*`second`*` [, `*`millisecond`*` ]]]]]);`

Arguments
`year` Un nombre à quatre chiffres, par exemple, 2000.

`month` Un entier entre 0 (janvier) et 11 (décembre).

`date` Un entier entre 1 et 31. Cet argument est optionnel.

`hour` Un entier entre 0 (minuit) et 23 (23h00).

`minute` Un entier entre 0 et 59. Cet argument est optionnel.

`second` Un entier entre 0 et 59. Cet argument est optionnel.

`millisecond` Un entier entre 0 et 999. Cet argument est optionnel.

Description
Méthode ; renvoie le nombre de millisecondes écoulées entre le premier janvier 1970 à minuit, temps universel, et la date spécifiée dans les arguments. Cette méthode est statique et elle est invoquée avec le constructeur d'objet Date et non avec un objet Date spécifique. Cette méthode vous permet de créer un objet Date qui adopte le temps universel, alors que le constructeur Date adopte l'heure locale.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant crée un nouvel objet Date `gary_birthday` défini en temps universel. Il s'agit de la variante en temps universel de l'exemple utilisé pour la méthode du constructeur `new Date()`.

```
gary_birthday = new Date(Date.UTC(1974, 7, 8));
```

delete

Syntaxe

```
delete (reference);
```

Arguments

reference Le nom de la variable ou de l'objet à supprimer.

Description

Opérateur ; détruit l'objet ou la variable spécifié comme *reference* et renvoie `true` si l'objet a été supprimé avec succès ; sinon, renvoie `false`. Cet opérateur est utile pour libérer la mémoire utilisée par les scripts. Bien que `delete` soit un opérateur, il est généralement utilisé comme une instruction :

```
delete x;
```

L'opérateur delete peut échouer et renvoyer false si la *reference* n'existe pas ou ne peut pas être effacée. Les objets et les propriétés prédéfinis et les variables déclarées avec `var` ne peuvent pas être supprimés.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant crée un objet, l'utilise et le supprime une fois qu'il n'est plus nécessaire.

```
account = new Object();
  account.name = 'Jon';
  account.balance = 10000;
  ...
  delete account;
```

L'exemple suivant supprime la propriété d'un objet.

```
// create the new object "account"

account = new Object();

// assign property name to the account

  account.name = 'Jon';

// delete the property

delete account.name;
```

L'exemple suivant montre une autre suppression de propriété d'objet.

```
// create an Array object with length 0
array = new Array();
// Array.length is now 1
   array[0] = "abc";
// add another element to the array,Array.length is now 2
   array[1] = "def";
// add another element to array,Array.length is now 3
   array[2] = "ghi";
// array[2] is deleted, but Array.length is not changed,
   delete array[2];
```

L'exemple suivant illustre le comportement de `delete` sur les références d'objets.

```
// create a new object, and assign the variable ref1 to refer to
the object

ref1 = new Object();

ref1.name = "Jody";

// copy the reference variable into a new variable, and delete
ref1

ref2 = ref1;

delete ref1;
```

Si `ref1` n'avait pas été copiée dans `ref2`, l'objet aurait été supprimé lorsque nous avons supprimé `ref1`, car il n'y aurait plus eu de référence à l'objet. Si nous détruisions `ref2`, il n'y aurait plus de référence à l'objet et il serait détruit et la mémoire qu'il utilisait serait à nouveau disponible.

Voir aussi

« var » à la page 387

do...while

Syntaxe

```
do {
statement;
} while (condition);
```

Arguments

condition La condition à évaluer.

statement L'instruction à exécuter tant que *condition* évalue `true`.

Description

Action ; exécute les instructions puis évalue la condition dans la boucle, tant que la condition est vraie.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« break » à la page 235
« continue » à la page 241

_droptarget

Syntaxe
draggableInstanceName._droptarget

Arguments
draggableInstanceName Le nom d'une occurrence de clip qui était la cible d'une action startDrag.

Description
Propriété (lecture seule) ; renvoie le chemin absolu (noté avec la syntaxe à barre oblique) de l'occurrence de clip sur laquelle *draggableInstanceName* était lancé. La propriété _droptarget renvoie toujours un chemin commençant par /. Pour comparer la propriété _droptarget d'une occurrence avec une référence, utilisez eval pour convertir la valeur renvoyée dans la syntaxe à barre oblique en référence.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant évalue la propriété _droptarget de l'occurrence de clip garbage et utilise eval pour la convertir d'une syntaxe à barre oblique en une référence avec syntaxe à point. La référence garbage est ensuite comparée avec la référence à l'occurrence de clip trash. Si les deux références sont identiques, la visibilité de garbage est définie sur false. Si elles ne sont pas identiques, l'occurrence garbage est redéfinie sur sa position d'origine.

```
if (eval(garbage._droptarget) == _root.trash) {
   garbage._visible = false;
} else {
   garbage._x = x_pos;
   garbage._y = y_pos;
}
```

Les variables x_pos et y_pos sont définies sur l'image 1 de l'animation avec le script suivant :

```
x_pos = garbage._x;
y_pos = garbage._y;
```

Voir aussi
« startDrag » à la page 368

duplicateMovieClip

Syntaxe

```
duplicateMovieClip(target, newname, depth);
```

Arguments

target Le chemin cible de l'animation à dupliquer.

newname Un identifiant unique pour le clip dupliqué.

depth Le niveau de profondeur du clip. Le niveau de profondeur est l'ordre d'empilage qui détermine l'apparition des clips et des objets lorsqu'ils se recouvrent. Le premier clip que vous créez, ou l'occurrence que vous faites glisser sur la scène, possède un niveau de profondeur 0. Vous devez affecter un niveau de profondeur différent à chaque clip successif ou dupliqué pour éviter qu'il ne remplace des animations à des niveaux déjà occupés ou le clip original.

Description

Action ; crée une occurrence d'un clip pendant la lecture de l'animation. La duplication de clips commence toujours à l'image 1, peu importe l'image sur laquelle se trouvait le clip original. Les variables du clip parent ne sont pas copiées dans le clip dupliqué. Si le clip parent est supprimé, le clip dupliqué l'est également. Utilisez l'action ou la méthode `removeMovieClip` pour supprimer une occurrence de clip créée avec `duplicateMovieClip`.

Lecteur

Flash 4 et versions suivantes.

Exemple

Cette instruction duplique l'occurrence du clip `flower` dix fois. La variable `i` est utilisée pour créer un nouveau nom d'occurrence et une profondeur.

```
on(release) {
   amount = 10;
   while(amount>0) {
        duplicateMovieClip (_root.flower, "mc" + i, i);
        setProperty("mc" + i, _x, random(275));
        setProperty("mc" + i, _y, random(275));
        setProperty("mc" + i, _alpha, random(275));
        setProperty("mc" + i, _xscale, random(50));
        setProperty("mc" + i, _yscale, random(50));
        i = i + 1;
        amount = amount-1;
    }
}
```

Voir aussi

« removeMovieClip » à la page 350
« MovieClip.removeMovieClip » à la page 324

else

Syntaxe
`else {statement(s)};`

Arguments
statement(s) Une série alternative d'instructions à effectuer si la condition spécifiée dans l'instruction `if` est `false`.

Description
Action ; spécifie les actions, clauses, arguments ou autres conditions à effectuer si l'instruction initiale `if` renvoie `false`.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« if » à la page 278

eq (égal–propre aux chaînes)

Syntaxe
expression1 eq *expression2*

Arguments
expression1, expression2 Nombres, chaînes ou variables.

Description
Opérateur de comparaison ; compare l'égalité de deux expressions et renvoie `true` si *expression1* est égale à *expression2* ; sinon, renvoie `false`.

Lecteur
Flash 1 et versions suivantes. Cet opérateur est désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur == (égalité) est recommandée.

Voir aussi
« == (égalité) » à la page 216

escape

Syntaxe
`escape(expression);`

Arguments
expression L'expression à convertir en chaîne et à coder dans un format de code URL.

Description
Fonction ; convertit l'argument en une chaîne et le code dans un format de code URL où tous les caractères alphanumériques sont échappés avec des séquences % hexadécimales.

Lecteur
Flash 5 et versions suivantes.

Exemple
```
escape("Hello{[World]}");
```
Le résultat de ce code est le suivant :

```
Hello%7B%5BWorld%5D%7D
```

Voir aussi
« unescape » à la page 384

eval

Syntaxe
```
eval(expression);
```

Arguments
expression Une chaîne contenant le nom d'une variable, d'une propriété, d'un objet ou d'un clip à extraire.

Description
Fonction ; accède aux variables, propriétés, objets ou clips par le nom. Si l'*expression* est une variable ou une propriété, la valeur de la variable ou de la propriété est renvoyée. Si l'*expression* est un objet ou un clip, une référence à l'objet ou au clip est renvoyée. Si l'élément nommé dans l'*expression* est introuvable, indéfini est renvoyé.

Dans Flash 4, la fonction `eval` était utilisée pour simuler un tableau ; dans Flash 5, il est recommandé d'utiliser l'objet Chaîne pour créer des tableaux.

Remarque : L'action ActionScript `eval` est différente de la fonction JavaScript `eval` et ne peut pas être utilisée pour évaluer des instructions.

Lecteur
Flash 5 et versions suivantes pour une pleine fonctionnalité. Vous pouvez utiliser `eval` lors d'exportation vers le Flash 4 Player, mais vous devez utiliser une notation à barre oblique et vous ne pouvez accéder qu'aux variables, pas aux propriétés ni aux objets.

Exemple

L'exemple suivant utilise `eval` pour déterminer la valeur de la variable `x` et la définit sur la valeur de `y`.

```
x = 3;
y = eval("x");
```

L'exemple suivant utilise eval pour faire référence à l'objet de clip associé à l'occurrence de clip sur la scène, `Ball`.

```
eval("_root.Ball");
```

Voir aussi

« Tableau (objet) » à la page 223

evaluate

Syntaxe

statement;

Arguments

Aucun.

Description

Action ; crée une nouvelle ligne vide et insère un ; pour entrer des instructions de script uniques en utilisant le champ Expression dans le panneau Actions. L'instruction `evaluate` permet également aux utilisateurs écrivant des scripts en mode Normal dans le panneau Actions de Flash 5 d'appeler des fonctions.

Lecteur

Flash 5 et versions suivantes.

_focusrect

Syntaxe

_focusrect = *Booléen*;

Arguments

Booléen `true` ou `false`.

Description

Propriété (globale) ; spécifie si un rectangle jaune apparaît autour du bouton courant ciblé. La valeur par défaut `true` (non-zéro) affiche un rectangle jaune autour du bouton ou du champ de texte actuellement ciblé au fur et à mesure que l'utilisateur appuie sur la touche de tabulation pour naviguer. Spécifiez `false` pour n'afficher que l'état « ciblé » du bouton (le cas échéant) lorsque les utilisateurs naviguent.

Lecteur

Flash 4 et versions suivantes.

for

Syntaxe

```
for(init; condition; next); {
statement;
}
```

Arguments

init Une expression à évaluer avant de commencer la séquence de boucle, généralement une expression d'affectation. Une instruction `var` est également autorisée pour cet argument.

condition Une condition qui évalue `true` ou `false`. La condition est évaluée avant chaque itération de boucle ; la boucle sort lorsque la condition évalue `false`.

next Une expression à évaluer après chaque itération de boucle ; généralement une expression d'affectation utilisant les opérateurs `++` (incrémentation) ou `--` (décrémentation).

statement Une instruction à l'intérieur du corps de la boucle à exécuter.

Description

Action ; une construction de boucle qui évalue l'expression `init` (initialiser) une fois, puis commence une séquence de boucle par laquelle, aussi longtemps que *condition* évalue `true`, *statement* est exécutée et l'expression suivante évaluée.

Certaines propriétés ne peuvent pas être énumérées par les actions `for` ou `for..in`. Par exemple, les méthodes intégrées de l'objet Chaîne (`Array.sort` et `Array.reverse`) ne sont pas comprises dans l'énumération d'un objet Chaîne et les propriétés de clip comme `_x` et `_y`, ne sont pas énumérées.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `for` pour additionner les éléments dans un tableau.

```
for(i=0; i<10; i++) {
array [i] = (i + 5)*10;
}
```

Renvoie le tableau suivant :

```
[50, 60, 70, 80, 90, 100, 110, 120, 130, 140]
```

L'exemple suivant utilise `for` pour effectuer la même action à maintes reprises. Dans le code ci-dessous, la boucle `for` additionne les nombres de 1 à 100.

```
var sum = 0;
  for (var i=1; i<=100; i++) {
    sum = sum + i;
  }
```

Voir aussi

« ++ (incrémentation) » à la page 189
« -- (décrémentation) » à la page 189
« for..in » à la page 268
« var » à la page 387

for..in

Syntaxe

```
for(variableiterant in object){
statement; }
```

Arguments

variableiterant Le nom d'une variable agissant comme un itérant, référençant chaque propriété d'un objet ou chaque élément d'un tableau.

object Le nom d'un objet devant être itéré.

statement Une instruction à exécuter pour chaque itération.

Description

Action ; effectue une boucle sur les propriétés d'un objet ou sur les éléments d'un tableau et exécute *statement* pour chaque propriété d'un objet.

Certaines propriétés ne peuvent pas être énumérées par les actions `for` ou `for..in`. Par exemple, les méthodes intégrées de l'objet Chaîne (`Array.sort` et `Array.reverse`) ne sont pas comprises dans l'énumération d'un objet Chaîne, et les propriétés de clip comme `_x` et `_y` ne sont pas énumérées.

La construction `for..in` itère sur les propriétés des objets dans la chaîne prototype de l'objet itéré. Si le prototype de l'enfant est parent, itérer sur les propriétés de l'enfant avec `for...in` itérera également sur les propriétés de parent.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise for..in pour itérer sur les propriétés d'un objet.

```
myObject = { name:'Tara', age:27, city:'San Francisco' };
for (name in myObject) {
  trace ("myObject." + name + " = " + myObject[name]);
}
```

Le résultat de cet exemple est le suivant :

```
myObject.name = Tara
myObject.age = 27
myObject.city = San Francisco
```

L'exemple suivant utilise l'opérateur typeof avec for..in pour itérer sur un enfant particulier.

```
for (name in myMovieClip) {
  if (typeof (myMovieClip[name]) = "movieclip") {
    trace ("I have a movie clip child named " + name);
  }
}
```

L'exemple suivant énumère les enfants d'un clip et envoie chacun dans l'image 2 de leurs scénarios respectifs. Le clip RadioButtonGroup est un parent de plusieurs enfants, _RedRadioButton_, _GreenRadioButton_ et _BlueRadioButton.

```
for (var name in RadioButtonGroup) {
  RadioButtonGroup[name].gotoAndStop(2);
}
```

_framesloaded

Syntaxe

instancename._framesloaded

Arguments

instancename Le nom d'une occurrence de clip devant être évaluée.

Description

Propriété (lecture seule) ; le nombre d'images qui ont été chargées depuis une animation en continu. Cette propriété est utile pour déterminer si le contenu d'une image spécifique, et de toutes les images avant elle, a été chargé et est disponible localement dans le navigateur d'un utilisateur. Cette propriété est utile pour contrôler le processus de téléchargement d'animations importantes. Par exemple, vous pourriez souhaiter afficher un message aux utilisateurs leur indiquant que l'animation est en chargement jusqu'à ce qu'une image spécifique dans l'animation soit chargée.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant utilise la propriété `_framesloaded` pour coordonner le départ de l'animation avec le nombre d'images chargées.

```
if (_framesloaded >= _totalframes) {
gotoAndPlay ("Scene 1", "start");
} else {
setProperty ("_root.loader", _xscale, (_framesloaded/
_totalframes)*100);
}
```

fscommand

Syntaxe
`fscommand(`*`command`*`, `*`arguments`*`);`

Arguments
`command` Une chaîne transmise à l'application hôte pour toute utilisation.

`arguments` Une chaîne transmise à l'application hôte pour toute utilisation.

Description
Action ; permet à l'animation Flash de communiquer avec le programme hôte du Flash Player. Dans un navigateur Internet, `fscommand` appelle la fonction JavaScript `moviename_Dofscommand` dans la page HTML contenant l'animation Flash, où `moviename` est le nom du Flash Player tel qu'il est affecté par l'attribut `NAME` de la balise `EMBED` ou par la propriété ID de la balise `OBJECT`. Si le nom `theMovie` a été affecté au Flash Player, la fonction JavaScript appelée est `theMovie_Dofscommand`.

Lecteur
Flash 3 et versions suivantes.

function

Syntaxe

```
function functionname ([argument0, argument1,...argumentN]){
statement(s)
}

function ([argument0, argument1,...argumentN]){
statement(s)
}
```

Arguments

functionname Le nom de la nouvelle fonction.

argument Zéro ou plusieurs chaînes, nombres ou objets à transmettre à `function`.

statements Zéro ou plusieurs instructions ActionScript définies pour le corps de `function`.

Description

Action ; un jeu d'instructions que vous définissez pour effectuer une certaine tâche. Vous pouvez *déclarer*, ou définir, une fonction dans un emplacement et l'appeler, ou l'invoquer, depuis différents scripts dans une animation. Lorsque vous définissez une fonction, vous pouvez également spécifier les arguments de cette fonction. Les arguments sont des supports pour les valeurs sur lesquelles la fonction opèrera. Vous pouvez transmettre à une fonction différents arguments, également appelés paramètres, chaque fois que vous l'appelez.

Utilisez l'action `return` dans les *statement(s)* d'une fonction pour obliger une fonction à renvoyer, ou générer, une valeur.

Emploi 1 : Déclare une `function` en spécifiant *functionname*, *arguments* et *statement(s)*. Lorsqu'une fonction est appelée, la déclaration de la fonction est invoquée. La référence en aval est autorisée ; au sein d'une même liste Action, une fonction peut être déclarée après avoir été appelée. Une déclaration de fonction remplace toute déclaration précédente de cette même fonction. Vous pouvez utiliser cette syntaxe partout où une instruction est autorisée.

Emploi 2 : Crée une fonction anonyme et la renvoie. Cette syntaxe est utilisée dans les expressions et est particulièrement utile pour installer des méthodes dans des objets.

Lecteur
Flash 5 et versions suivantes.

Exemple
(Emploi 1) L'exemple suivant définit la fonction `sqr`, qui accepte un argument et renvoie `square(x*x)` de l'argument. À noter que si la fonction est déclarée et utilisée dans le même script, la déclaration de la fonction doit apparaître après avoir utilisé la fonction.

```
y=sqr(3);
function sqr(x) {
return x*x;
}
```

(Emploi 2) La fonction suivante définit un objet Circle :

```
function Circle(radius) {
  this.radius = radius;
}
```

L'instruction suivante définit une fonction anonyme qui calcule l'aire d'un cercle et l'associe à l'objet `Circle` comme méthode :

```
Circle.prototype.area = function () {return Math.PI * this.radius
* this.radius}
```

ge (supérieur ou égal à–propre aux chaînes)

Syntaxe
expression1 ge *expression2*

Arguments
expression1, expression2 Nombres, chaînes ou variables.

Description
Opérateur (comparaison) ; compare *expression1* avec *expression2* et renvoie `true` si *expression1* est supérieure ou égale à *expression2* ; sinon, renvoie `false`.

Lecteur
Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur >= est recommandée.

Voir aussi
« >= (supérieur ou égal à) » à la page 217

getProperty

Syntaxe

```
getProperty(instancename, property);
```

Arguments

instancename Le nom d'occurrence d'un clip pour lequel la propriété est extraite.

property Une propriété d'un clip, comme, par exemple, les coordonnées *x* ou *y*.

Description

Fonction ; renvoie la valeur de la *property* spécifiée pour l'occurrence du clip.

Lecteur

Flash 4 et versions ultérieures.

Exemple

L'exemple suivant extrait la coordonnée de l'axe horizontal (_x) pour le clip myMovie.

```
getProperty(_root.myMovie_item._x);
```

getTimer

Syntaxe

```
getTimer();
```

Arguments

Aucun.

Description

Fonction ; renvoie le nombre de millisecondes écoulées depuis le démarrage de la lecture de l'animation.

Lecteur

Flash 4 et versions suivantes.

getURL

Syntaxe

```
getURL(url [, window [, variables]]);
```

Arguments

url L'URL où se trouve le document à obtenir. L'URL doit se trouver dans le même sous-domaine que l'URL où se trouve l'animation en cours.

window Un argument optionnel spécifiant la fenêtre ou l'image HTML dans laquelle le document doit être chargé. Entrez le nom d'une fenêtre spécifique ou choisissez parmi les noms cibles réservés suivants :

- `_self` spécifie l'image courante dans la fenêtre courante.
- `_blank` spécifie une nouvelle fenêtre.
- `_parent` spécifie le parent de l'image courante.
- `_top` spécifie l'image de haut niveau de la fenêtre courante.

variables Un argument optionnel spécifiant une méthode pour envoyer les variables. S'il n'y a pas de variables, passez cet argument ; sinon, spécifiez si les variables doivent être chargées avec la méthode `GET` ou `POST`. `GET` ajoute les variables à la fin de l'URL et est utilisée pour des petits nombres de variables. `POST` envoie les variables dans un en-tête HTTP séparé et est utilisée pour les longues chaînes de variables.

Description

Action ; charge un document depuis une URL spécifique dans une fenêtre ou transmet les variables à une autre application à une URL définie. Pour tester cette action, assurez-vous que le fichier à charger se trouve dans l'emplacement spécifié. Pour utiliser une URL absolue (par exemple, *http://www.myserver.com*), il vous faut une connexion de réseau.

Lecteur

Flash 2 et versions suivantes. Les options GET et POST ne sont disponibles que dans Flash 4 et les versions suivantes du Player.

Exemple

Cet exemple charge une nouvelle URL dans une fenêtre de navigation vide. L'action `getURL` cible la variable `incomingAd` comme paramètre *`url`* de sorte que vous pouvez changer l'URL chargée sans avoir à éditer l'animation Flash. La valeur de la variable `incomingAd` est transmise à Flash plus tôt dans l'animation avec une action `loadVariables`.

```
on(release) {
    getURL(incomingAd, "_blank");
}
```

Voir aussi

« loadVariables » à la page 295
« XML.send » à la page 407
« XML.sendAndLoad » à la page 407
« XMLSocket.send » à la page 417

getVersion

Syntaxe

```
getVersion();
```

Arguments

Aucun.

Description

Fonction ; renvoie une chaîne contenant la version Flash Player et les informations de plate-forme.

Cette fonction ne marche pas en mode de test d'animation et ne renverra que les informations concernant les versions 5 et suivantes du Flash Player.

Exemple

L'exemple suivant montre une chaîne renvoyée par la fonction `getVersion` :

```
WIN 5,0,17,0
```

Cette chaîne indique que la plate-forme est Windows et que le numéro de version du Flash Player est version supérieure 5, version inférieure 17(5.17).

Lecteur

Flash 5 et versions suivantes.

gotoAndPlay

Syntaxe

```
gotoAndPlay(scene, frame);
```

Arguments

scene Le nom de la scène dans laquelle la tête de lecture est envoyée.

frame Le numéro de l'image dans laquelle la tête de lecture est envoyée.

Description

Action ; envoie la tête de lecture dans une image spécifiée d'une scène et lit à partir de cette image. Si aucune scène n'est spécifiée, la tête de lecture va dans l'image spécifiée de la scène courante.

Lecteur

Flash 2 et versions suivantes.

Exemple

Lorsque l'utilisateur clique sur un bouton auquel est affectée l'action `gotoAndPlay`, la tête de lecture est envoyée à l'image 16 et commence à lire.

```
on(release) {
    gotoAndPlay(16);
}
```

gotoAndStop

Syntaxe

```
gotoAndStop(scene, frame);
```

Arguments

scene Le nom de la scène dans laquelle la tête de lecture est envoyée.

frame Le numéro de l'image dans laquelle la tête de lecture est envoyée.

Description

Action ; envoie la tête de lecture dans l'image spécifiée de la scène et l'arrête. Si aucune scène n'est spécifiée, la tête de lecture est envoyée dans l'image de la scène courante.

Lecteur

Flash 2 et versions suivantes.

Exemple

Lorsque l'utilisateur clique sur un bouton auquel l'action `gotoAndStop` est affectée, la tête de lecture est envoyée dans l'image 5 et la lecture de l'animation cesse.

```
on(release) {
    gotoAndStop(5);
}
```

gt (supérieur à–propre aux chaînes)

Syntaxe

expression1 gt *expression2*

Arguments

expression1, expression2 Nombres, chaînes ou variables.

Description

Opérateur (comparaison) ; compare *expression1* avec *expression2* et renvoie `true` si *expression1* est supérieure à *expression2* ; sinon, renvoie `false`.

Lecteur

Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur > est recommandée.

Voir aussi

« > (supérieur à) » à la page 216

_height

Syntaxe

instancename._height
instancename._height = *value*;

Arguments

instancename Un nom d'occurrence de clip pour lequel la propriété `_height` doit être définie ou extraite.

value Un entier spécifiant la hauteur de l'animation en pixels.

Description

Propriété ; définit et extrait la hauteur de l'espace occupé par le contenu d'une animation. Dans les versions précédentes de Flash, `_height` et `_width` étaient des propriétés en lecture seule ; dans Flash 5, ces propriétés peuvent être définies.

Lecteur

Flash 4 et versions suivantes.

Exemple

L'exemple de code suivant définit la hauteur et la largeur d'un clip lorsque l'utilisateur clique sur la souris.

```
onClipEvent(mouseDown) {
   _width=200;
   _height=200;
}
```

_highquality

Syntaxe

```
_highquality = value;
```

Arguments

value Le niveau d'antialias appliqué à l'animation. Spécifiez 2 (BEST) pour appliquer une haute qualité avec le lissage bitmap toujours actif. Spécifiez 1 (qualité élevée) pour appliquer l'antialias ; cela lissera les bitmaps si le clip ne contient pas d'animation. Spécifiez 0 (faible qualité) pour empêcher l'antialias.

Description

Propriété (globale) ; spécifie le niveau d'antialias appliqué à l'animation en cours.

Lecteur

Flash 4 et versions suivantes.

Voir aussi

« _quality » à la page 349
« toggleHighQuality » à la page 382

if

Syntaxe

```
if(condition) {

statement;

{
```

Arguments

conditional Une expression qui évalue `true` ou `false`. Par exemple, `if(name == "Erica")`, évalue la variable `name` pour voir s'il s'agit de “Erica”.

statements Les instructions à exécuter si, ou quand, la condition évalue `true`.

Description

Action ; évalue une condition pour déterminer la prochaine action dans une animation. Si la condition est `true`, Flash exécute l'instruction qui suit. Utilisez `if` pour créer une logique arborescente dans vos scripts.

Lecteur

Flash 4 et versions suivantes.

Voir aussi

« else » à la page 264
« for » à la page 267
« for..in » à la page 268

ifFrameLoaded

Syntaxe

```
ifFrameLoaded(scene, frame) {
statement;}
```

```
ifFrameLoaded(frame) {
statement;}
```

Arguments

scene La scène qui est interrogée.

frame Le numéro de l'image ou l'étiquette de l'image à charger avant l'exécution de la prochaine instruction.

Description

Action ; vérifie si le contenu d'une image est disponible localement. Utilisez `ifFrameLoaded` pour commencer la lecture d'une animation simple pendant que le reste de l'animation est téléchargé sur un ordinateur local. La différence entre l'utilisation de `_framesloaded` et `ifFrameLoaded` réside dans le fait que `_framesloaded` vous permet d'ajouter des instructions `if` ou `else`, alors que l'action `ifFrameLoaded` vous permet de spécifier un nombre d'images dans une instruction simple.

Lecteur

Flash 3 et versions suivantes. L'action `ifFrameLoaded` est désapprouvée dans Flash 5 ; l'utilisation de l'action `_framesloaded` est encouragée.

Voir aussi

« _framesloaded » à la page 269

#include

Syntaxe

```
#include "filename.as";
```

Arguments

filename.as Le nom de fichier à inclure ; `.as` est l'extension de fichier recommandée.

Description

Action ; inclut le contenu du fichier spécifié dans l'argument lorsque l'animation est testée, éditée ou exportée. L'action `#include` est invoquée lorsque vous testez, éditez ou exportez. L'action `#include` est vérifiée lorsqu'un contrôle de syntaxe survient.

Lecteur

N/A

Infinity

Syntaxe
```
Infinity
```

Arguments
Aucun.

Description
Variable générale ; une variable prédéfinie avec la valeur ECMA-262 comme infini.

Lecteur
Flash 5 et versions suivantes.

int

Syntaxe
```
int(value);
```

Arguments
value Un nombre devant être arrondi en entier.

Description
Fonction ; convertit un nombre décimal à la valeur de l'entier le plus proche.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5 ; l'utilisation de la méthode `Math.floor` est recommandée.

Voir aussi
« Math.floor » à la page 302

isFinite

Syntaxe
```
isFinite(expression);
```

Arguments
expression Le booléen, la variable ou une autre expression à évaluer.

Description
Fonction de haut niveau ; évalue l'argument et renvoie `true` s'il s'agit d'un nombre fini et `false` s'il s'agit d'infini ou d'infini négatif. La présence d'infini, ou d'infini négatif, indique une condition d'erreur mathématique (une division par 0, par exemple).

Lecteur
Flash 5 et versions suivantes.

Exemple
Les exemples suivants renvoient des valeurs pour `isFinite` :

`isFinite(56)` renvoie `true`

`isFinite(Number.POSITIVE_INFINITY)` renvoie `false`

`isNaN(Number.POSITIVE_INFINITY)` renvoie `false`

isNaN

Syntaxe
`isNaN(`*`expression`*`);`

Arguments
`expression` Le booléen, la variable ou une autre expression à évaluer.

Description
Fonction de haut niveau ; évalue l'argument et renvoie `true` si la valeur n'est pas un nombre (`NaN`), indiquant la présence d'erreurs mathématiques.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant illustre la valeur renvoyée pour `isNan` :

`isNan("Tree")` renvoie `true`

`isNan(56)` renvoie `false`

`isNaN(Number.POSITIVE_INFINITY)` renvoie `false`

Key (objet)

L'objet Key est un objet de haut niveau auquel vous pouvez accéder sans utiliser de constructeur. Utilisez les méthodes de l'objet Key pour construire une interface qui pourra être contrôlée par un utilisateur possédant un clavier standard. Les propriétés de l'objet Key sont des constantes représentant les touches les plus souvent utilisées pour contrôler les jeux. Voir Annexe B, « Touches de clavier et valeurs des codes-clés », pour une liste complète des valeurs des codes-clés.

Exemple

```
onClipEvent (enterFrame) {
    if(Key.isDown(Key.RIGHT)) {
        setProperty ("", _x, _x+10);
    }
}
or
onClipEvent (enterFrame) {
    if(Key.isDown(39)) {
        setProperty("", _x, _x+10);
    }
}
```

Tableau des méthodes de l'objet Key

Méthode	Description
`getAscii;`	Renvoie la valeur ASCII de la dernière touche enfoncée.
`getCode;`	Renvoie le code-clé virtuel de la dernière touche enfoncée.
`isDown;`	Renvoie `true` si la touche spécifiée dans l'argument est enfoncée.
`isToggled;`	Renvoie `true` si les touches Verr Num ou Caps Lock sont activées.

Tableau des propriétés de l'objet Key

Toutes les propriétés de l'objet Key sont des constantes.

Propriété	Description
`BACKSPACE`	Constante associée avec la valeur de code-clé pour la touche de rappel arrière (9).
`CAPSLOCK`	Constante associée avec la valeur de code-clé pour la touche Caps Lock (20).
`CONTROL`	Constante associée avec la valeur de code-clé pour la touche Ctrl (17).

Propriété	Description
DELETEKEY	Constante associée avec la valeur de code-clé pour la touche Suppr (46).
DOWN	Constante associée avec la valeur de code-clé pour la touche de flèche vers le bas (40).
END	Constante associée avec la valeur de code-clé pour la touche Fin (35).
ENTER	Constante associée avec la valeur de code-clé pour la touche Entrée (13).
ESCAPE	Constante associée avec la valeur de code-clé pour la touche Échap (27).
HOME	Constante associée avec la valeur de code-clé pour la touche début d'écran (36).
INSERT	Constante associée avec la valeur de code-clé pour la touche Inser (45).
LEFT	Constante associée avec la valeur de code-clé pour la touche flèche vers la gauche (37).
PGDN	Constante associée avec la valeur de code-clé pour la touche de défilement vers le bas (34).
PGUP	Constante associée avec la valeur de code-clé pour la touche de défilement vers le haut (33).
RIGHT	Constante associée avec la valeur de code-clé pour la touche flèche vers la droite (39).
SHIFT	Constante associée avec la valeur de code-clé pour la touche Maj (16).
SPACE	Constante associée avec la valeur de code-clé pour la barre d'espace (32).
TAB	Constante associée avec la valeur de code-clé pour la touche Tab (9).
UP	Constante associée avec la valeur de code-clé pour la touche flèche vers le haut (38).

Key.BACKSPACE

Syntaxe
```
Key.BACKSPACE
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche de rappel arrière (9).

Lecteur
Flash 5 et versions suivantes.

Key.CAPSLOCK

Syntaxe
```
Key.CAPSLOCK
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Caps Lock (20).

Lecteur
Flash 5 et versions suivantes.

Key.CONTROL

Syntaxe
```
Key.CONTROL
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Ctrl (17).

Lecteur
Flash 5 et versions suivantes.

Key.DELETEKEY

Syntaxe
```
Key.DELETEKEY
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Suppr (46).

Lecteur
Flash 5 et versions suivantes.

Key.DOWN

Syntaxe
```
Key.DOWN
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche flèche vers le bas (40).

Lecteur
Flash 5 et versions suivantes.

Key.END

Syntaxe
```
Key.END
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Fin (35).

Lecteur
Flash 5 et versions suivantes.

Key.ENTER

Syntaxe
```
Key.ENTER
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Entrée (13).

Lecteur
Flash 5 et versions suivantes.

Key.ESCAPE

Syntaxe
```
Key.ESCAPE
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Échap (27).

Lecteur
Flash 5 et versions suivantes.

Key.getAscii

Syntaxe
```
Key.getAscii();
```

Arguments
Aucun.

Description
Méthode ; renvoie le code ASCII de la dernière touche enfoncée ou relâchée.

Lecteur
Flash 5 et versions suivantes.

Key.getCode

Syntaxe
```
Key.getCode();
```

Arguments
Aucun.

Description
Méthode ; renvoie la valeur de code-clé de la dernière touche enfoncée. Utilisez les informations de l'Annexe B, « Touches de clavier et valeurs des codes-clés », pour faire correspondre la valeur de code-clé renvoyée avec la touche virtuelle sur un clavier standard.

Lecteur
Flash 5 et versions suivantes.

Key.HOME

Syntaxe
```
Key.HOME
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche début d'écran (36).

Lecteur
Flash 5 et versions suivantes.

Key.INSERT

Syntaxe
```
Key.INSERT
```

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Inser (45).

Lecteur
Flash 5 et versions suivantes.

Key.isDown

Syntaxe
`Key.isDown(keycode);`

Arguments
keycode La valeur de code-clé affectée à une touche spécifique ou la propriété d'un objet Key associée à une touche spécifique. L'Annexe B, " Touches de clavier et valeurs des codes-clés », répertorie tous les codes-clés associés avec les touches d'un clavier standard.

Description
Méthode ; renvoie `true` si la touche spécifiée dans *keycode* est enfoncée. Sur Macintosh, les valeurs de codes-clés pour les touches Caps Lock et Verr Num sont identiques.

Lecteur
Flash 5 et versions suivantes.

Key.isToggled

Syntaxe
`Key.isToggled(keycode)`

Arguments
keycode Le code-clé pour Caps Lock (20) ou Verr Num (144).

Description
Méthode ; renvoie `true` si la touche Caps Lock ou Verr Num est activée (basculée). Sur Macintosh, les codes-clés de ces valeurs sont identiques.

Lecteur
Flash 5 et versions suivantes.

Key.LEFT

Syntaxe
`Key.LEFT`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche flèche vers la gauche (37).

Lecteur
Flash 5 et versions suivantes.

Key.PGDN

Syntaxe
`Key.PGDN`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche défilement vers le bas(34).

Lecteur
Flash 5 et versions suivantes.

Key.PGUP

Syntaxe
`Key.PGUP`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche défilement vers le haut (33).

Lecteur
Flash 5 et versions suivantes.

Key.RIGHT

Syntaxe
`Key.RIGHT`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche flèche vers la droite (39).

Lecteur
Flash 5 et versions suivantes.

Key.SHIFT

Syntaxe
`Key.SHIFT`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Maj (16).

Lecteur
Flash 5 et versions suivantes.

Key.SPACE

Syntaxe
`Key.SPACE`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la barre d'espace (32).

Lecteur
Flash 5 et versions suivantes.

Key.TAB

Syntaxe
`Key.TAB`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche Tab (9).

Lecteur
Flash 5 et versions suivantes.

Key.UP

Syntaxe
`Key.UP`

Arguments
Aucun.

Description
Propriété ; constante associée avec la valeur de code-clé pour la touche flèche vers le haut (38).

Lecteur
Flash 5 et versions suivantes.

le (inférieur ou égal à–propre aux chaînes)

Syntaxe
expression1 `le` *expression2*

Arguments
expression1, expression2 Nombres, chaînes ou variables.

Description
Opérateur (comparaison) ; compare *expression1* avec *expression2* et renvoie `true` si *expression1* est inférieure ou égale à *expression2* ; sinon, renvoie `false`.

Lecteur
Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur `<=` est recommandée.

Voir aussi
« <= (inférieur ou égal à) » à la page 213

length

Syntaxe
`length(`*expression*`);`

`length(`*variable*`);`

Arguments
expression Toute chaîne.

variable Le nom d'une variable.

Description
Fonction de chaîne ; renvoie la longueur de la chaîne spécifiée ou de la variable.

Lecteur
Flash 4 et versions suivantes. Cette fonction, ainsi que toutes les fonctions de chaîne, a été désapprouvée dans Flash 5. Il est recommandé d'utiliser les méthodes et la propriété `length` de l'objet Chaîne pour effectuer les mêmes opérations.

Exemple
L'exemple suivant renvoie la valeur de la chaîne `Hello`:

```
length("Hello")
```

Le résultat est 5.

Voir aussi
« " " (délimiteur de chaîne) » à la page 371
« String.length » à la page 375

_level

Syntaxe
```
_levelN;
```

Arguments
N Un entier non négatif spécifiant le niveau de profondeur. Par défaut, `_level` est défini sur 0, l'animation à la base de la hiérarchie.

Description
Propriété ; une référence au scénario de l'animation racine de *levelN*. Vous devez charger les animations en utilisant l'action `loadMovie` avant de les cibler avec la propriété `_level`.

Dans le Flash Player, un numéro est affecté aux animations en fonction de l'ordre dans lequel elles ont été chargées. L'animation qui a été chargée en premier est chargée au niveau inférieur, le niveau 0. L'animation du niveau 0 définit le taux d'images, la couleur d'arrière plan et la taille des images pour toutes les animations chargées par la suite. Les animations sont ensuite empilées en niveaux supérieurs numérotés au-dessus de l'animation du niveau 0. Le niveau où se trouve un clip est également appelé niveau de profondeur ou profondeur.

Lecteur
Flash 4 et versions suivantes.

Exemple

L'exemple suivant arrête le scénario de l'animation du niveau 0.

```
_level0.stop();
```

L'exemple suivant envoie le scénario de l'animation du niveau 4 vers l'image 5. L'animation du niveau 4 doit d'abord avoir été chargée avec une action `loadMovie`.

```
_level4.gotoAndStop(5);
```

Voir aussi

« loadMovie » à la page 293
« MovieClip.swapDepths » à la page 325

loadMovie

Syntaxe

```
loadMovie(url [,location/target, variables]]);
```

Arguments

`url` Une URL absolue ou relative pour le fichier SWF à charger. Un chemin relatif doit être relatif à SWF. L'URL doit se trouver dans le même sous-domaine que l'URL où se trouve l'animation courante. Pour une utilisation dans le Flash Player ou pour un test en mode test d'animation dans l'environnement de programmation de Flash, tous les fichiers SWF doivent être stockés dans le même dossier et les noms de fichiers ne doivent pas inclure de spécifications de dossier ni d'unité de disque.

`target` Un argument optionnel spécifiant un clip cible qui sera remplacé par l'animation chargée. L'animation chargée hérite des propriétés de position, de rotation et d'échelle du clip ciblé. Spécifier *`target`* est identique à la spécification de *`location`* (niveau) d'une animation cible. Vous ne devez pas spécifier les deux.

`location` Un argument optionnel spécifiant le niveau dans lequel l'animation est chargée. L'animation chargée hérite des propriétés de position, de rotation et d'échelle du clip ciblé. Pour charger la nouvelle animation en plus des animations existantes, spécifiez un niveau qui n'est pas occupé par une autre animation. Pour remplacer une animation existante par l'animation chargée, spécifiez un niveau occupé actuellement par une autre animation. Pour remplacer l'animation d'origine et vider les autres niveaux, chargez la nouvelle animation au niveau 0. L'animation du niveau 0 définit le taux d'images, la couleur d'arrière plan et la taille des images pour toutes les autres animations chargées.

variables Un argument optionnel spécifiant une méthode pour envoyer les variables associées à l'animation à charger. Les arguments doivent être les chaînes « GET » ou « POST ». S'il n'y a pas de variables, passez cet argument ; sinon, spécifiez si les variables doivent être chargées avec une méthode `GET` ou `POST`. `GET` ajoute les variables à la fin de l'URL et est utilisée pour des petits nombres de variables. `POST` envoie les variables dans un en-tête HTTP séparé et est utilisée pour les longues chaînes de variables.

Description

Action ; lit des animations supplémentaires sans fermer le Flash Player. Normalement, le Flash Player affiche une seule animation Flash Player (fichier SWF) puis se ferme. L'action `loadMovie` vous permet d'afficher plusieurs animations en une fois ou de basculer entre les animations sans charger un autre document HTML.

Vous pouvez charger des animations dans des niveaux qui possèdent déjà des fichiers SWF chargés. Si vous le faites, la nouvelle animation remplacera le fichier SWF existant. Si vous chargez une nouvelle animation dans le niveau 0, tous les niveaux sont vidés et le niveau 0 est remplacé par le nouveau fichier. Utilisez l'action `loadVariables` pour conserver l'animation active et pour mettre à jour les variables avec de nouvelles valeurs.

Utilisez l'action `unloadMovie` pour supprimer les animations chargées avec l'action`loadMovie` .

Lecteur

Flash 3 et versions suivantes.

Exemple

Cette instruction `loadMovie` est associée à un bouton de navigation intitulé Produits. Un clip invisible se trouve sur la scène et porte le nom d'occurrence `dropZone`. L'action loadMovie utilise ce clip comme paramètre cible pour charger products du fichier SWF dans la position correcte sur la scène.

```
on(release) {
    loadMovie("products.swf",_root.dropZone);
}
```

Voir aussi

« unloadMovie » à la page 385
« _level » à la page 292

loadVariables

Syntaxe

```
loadVariables (url,location [, variables]);
```

Arguments

url Une URL absolue ou relative où se trouvent les variables. L'hôte de l'URL doit se trouver dans le même sous-domaine que l'animation si l'accès se fait depuis un navigateur Internet.

location Un niveau ou une cible pour recevoir les variables. Dans le Flash Player, un numéro est affecté aux fichiers d'animation en fonction de l'ordre dans lequel ils ont été chargés. La première animation est chargée dans le niveau inférieur (niveau 0). Dans l'action `loadMovie`, vous devez spécifier un numéro de niveau pour chaque animation successive. Cet argument est optionnel.

variables Un argument optionnel spécifiant une méthode pour envoyer les variables. S'il n'y a pas de variables, passez cet argument ; sinon, spécifiez si les variables doivent être chargées avec une méthode `GET` ou `POST`. `GET` ajoute les variables à la fin de l'URL et est utilisée pour des petits nombres de variables. `POST` envoie les variables dans un en-tête HTTP séparé et est utilisée pour les longues chaînes de variables.

Description

Action ; lit les données depuis un fichier externe, comme un fichier texte ou un texte généré par un script CGI, Active Server Pages (ASP), ou Personal Home Page (PHP), et définit les valeurs des variables dans une animation ou un clip. Cette action peut également être utilisée pour mettre à jour les variables de l'animation active avec des nouvelles valeurs.

Le texte de l'URL spécifiée doit être au format standard MIME *application x www-urlformencoded* (un format standard utilisé par les scripts CGI). L'animation et les variables à charger doivent se trouver dans le même sous-domaine. Tout nombre de variables peut être spécifié. Par exemple, la séquence ci-dessous définit plusieurs variables :

```
company=Macromedia&address=600+Townsend&city=
San+Francisco&zip=94103
```

Lecteur

Flash 4 et versions suivantes.

Exemple
Cet exemple charge les informations depuis un fichier texte vers des champs de texte dans le scénario principal (niveau 0). Les noms de variables des champs de texte doivent correspondre aux noms de variables du fichier data.txt.

```
on(release) {
    loadVariables("data.txt", 0);
}
```

Voir aussi
« getURL » à la page 274
« MovieClip.loadMovie » à la page 320
« MovieClip.loadVariables » à la page 321

lt (inférieur à–propre aux chaînes)

Syntaxe
expression1 lt *expression2*

Arguments
expression1, expression2 Nombres, chaînes ou variables.

Description
Opérateur (comparaison) ; compare *expression1* avec *expression2* et renvoie `true` si *expression1* est inférieure à *expression2* ; sinon, renvoie `false`.

Lecteur
Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur < (inférieur à) est recommandée.

Voir aussi
« < (inférieur à) » à la page 210

Math (objet)

L'objet Math est un objet de niveau supérieur auquel vous pouvez accéder sans utiliser de constructeur.

Utilisez les méthodes et les propriétés de cet objet pour accéder aux constantes et aux fonctions mathématiques et pour les manipuler. Toutes les méthodes et les propriétés de l'objet Math sont statiques et doivent être appelées en utilisant la syntaxe `Math.method(argument)` ou `Math.constant`. Dans ActionScript, les constantes sont définies avec le maximum de précision des nombres à virgule flottante double précision IEEE-754.

L'objet Math est pleinement supporté par le Flash 5 Player. Dans le Flash 4 Player, les méthodes de l'objet Math fonctionnent, mais elles sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Plusieurs méthodes de l'objet Math utilisent le radian d'un angle comme argument. Vous pouvez utiliser l'équation ci-dessous pour calculer les valeurs en radians, ou simplement transmettre l'équation (en entrant une valeur pour les degrés) pour l'argument radian.

Pour calculer une valeur en radians, utilisez cette formule :

```
radian = Math.PI/180 * degree
```

L'exemple suivant transmet l'équation comme argument pour calculer le sinus d'un angle à 45 degrés :

`Math.SIN(Math.PI/180 * 45)` est identique à `Math.SIN(.7854)`

Tableau des méthodes de l'objet Math

Méthode	Description
`abs`	Calcule une valeur absolue.
`acos`	Calcule le cosinus d'un arc.
`asin`	Calcule le sinus d'un arc.
`atan`	Calcule la tangente d'un arc.
`atan2`	Calcule un angle depuis l'axe des x jusqu'au point.
`ceil`	Arrondit un nombre à l'entier supérieur le plus proche.
`cos`	Calcule un cosinus.
`exp`	Calcule une valeur exponentielle.
`floor`	Arrondit un nombre à l'entier inférieur le plus proche.
`log`	Calcule un logarithme naturel.
`max`	Renvoie le plus grand des deux entiers.
`min`	Renvoie le plus petit des deux entiers.
`pow`	Calcule *x* élevé à la puissance *y*.
`random`	Renvoie un nombre pseudo-aléatoire entre 0,0 et 1,0.
`round`	Arrondit au plus proche entier.
`sin`	Calcule un sinus.
`sqrt`	Calcule une racine carrée.
`tan`	Calcule une tangente.

Tableau des propriétés de l'objet Math

Toutes les propriétés de l'objet Math sont des constantes.

Propriété	Description
`E`	Constante d'Euler et la base des logarithmes naturels (approximativement 2,718).
`LN2`	Le logarithme naturel de 2 (approximativement 0,693).
`LOG2E`	Le logarithme de base 2 de *e* (approximativement 1,442).
`LN10`	Le logarithme naturel de 10 (approximativement 2,302).
`LOG10E`	Le logarithme de base 10 de *e* (approximativement 0,434).
`PI`	Le rapport de la circonférence d'un cercle à son diamètre (approximativement 3,14159).
`SQRT1_2`	La réciproque de la racine carrée de1/2 (approximativement 0,707).
`SQRT2`	La racine carrée de 2 (approximativement 1,414).

Math.abs

Syntaxe

```
Math.abs(x);
```

Arguments

x Tout nombre.

Description

Méthode ; calcule et renvoie une valeur absolue pour le nombre spécifié par l'argument *x*.

Lecteur

Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.acos

Syntaxe

```
Math.acos(x);
```

Arguments

x Un nombre entre –1,0 et 1,0.

Description

Méthode ; calcule et renvoie le cosinus de l'arc du nombre spécifié dans l'argument *x*, en radians.

Lecteur

Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.asin

Syntaxe

```
Math.asin(x);
```

Arguments

x Un nombre entre –1,0 et 1,0.

Description

Méthode ; calcule et renvoie le sinus de l'arc du nombre spécifié dans l'argument *x*, en radians.

Lecteur

Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.atan

Syntaxe
```
Math.atan(x);
```

Arguments
x Tout nombre.

Description
Méthode ; calcule et renvoie la tangente de l'arc pour le nombre spécifié dans l'argument *x*. La valeur renvoyée se situe entre pi négatif divisé par 2 et pi positif divisé par 2.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.atan2

Syntaxe
```
Math.atan2(y, x);
```

Arguments
x Un nombre spécifiant la coordonnée *x* du point.

y Un nombre spécifiant la coordonnée *y* du point.

Description
Méthode ; calcule et renvoie la tangente de l'arc *y*/*x* en radians. La valeur renvoyée représente l'angle opposé d'un triangle droit, où *x* est la longueur du côté adjacent et *y* est la longueur du côté opposé.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.ceil

Syntaxe
```
Math.ceil(x);
```

Arguments
x Un nombre ou une expression.

Description
Méthode ; renvoie le plafond du nombre ou de l'expression spécifiés. Le plafond d'un nombre est l'entier le plus proche supérieur ou égal à ce nombre.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.cos

Syntaxe
```
Math.cos(x);
```

Arguments
x Un angle mesuré en radians.

Description
Méthode ; renvoie le cosinus (une valeur entre –1,0 et 1,0) de l'angle spécifié par l'argument *x*. L'angle *x* doit être spécifié en radians. Utilisez les informations indiquées dans l'introduction de l'objet Math pour calculer un radian.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.E

Syntaxe
```
Math.E
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour la base des logarithmes naturels, exprimée sous la forme *e*. La valeur approximative de *e* est 2,71828.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.exp

Syntaxe
```
Math.exp(x);
```

Arguments
x L'exposant ; un nombre ou une expression.

Description
Méthode ; renvoie la valeur de la base du logarithme naturel (*e*), à la puissance de l'exposant spécifié dans l'argument *x*. La constante `Math.E` peut fournir la valeur de *e*.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.floor

Syntaxe
```
Math.floor(x);
```

Arguments
x Un nombre ou une expression.

Description
Méthode ; renvoie le plancher du nombre ou de l'expression spécifiés dans l'argument *x*. Le plancher est l'entier le plus proche inférieur ou égal au nombre ou à l'expression spécifiés.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Exemple
L'exemple suivant renvoie une valeur de 12.

```
Math.floor(12.5);
```

Math.log

Syntaxe
```
Math.log(x);
```

Arguments
x Un nombre ou une expression avec une valeur supérieure à 0.

Description
Méthode ; renvoie le logarithme naturel de l'argument *x*.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.LOG2E

Syntaxe
```
Math.LOG2E
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour le logarithme de base 2 de la constante *e* (`Math.E`), exprimée sous la forme $\log_e 2$, avec une valeur approximative de 1,442695040888963387.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.LOG10E

Syntaxe
```
Math.LOG10E
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour le logarithme de base 10 de la constante *e* (`Math.E`), exprimée sous la forme $\log_{10} e$, avec une valeur approximative de 0,43429448190325181667.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.LN2

Syntaxe
```
Math.LN2
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour le logarithme naturel de 2, exprimée sous la forme $\log_e 2$, avec une valeur approximative de 0,69314718055994528623.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.LN10

Syntaxe
```
Math.LN10
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour le logarithme naturel de 10, exprimée sous la forme $\log_e 10$, avec une valeur approximative de 2,30258509299404459011.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.max

Syntaxe
```
Math.max(x, y);
```

Arguments
x Un nombre ou une expression.

y Un nombre ou une expression.

Description
Méthode ; évalue *x* et *y* et renvoie la valeur la plus grande.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.min

Syntaxe
```
Math.min(x, y);
```

Arguments
x Un nombre ou une expression.

y Un nombre ou une expression.

Description
Méthode ; évalue *x* et *y* et renvoie la valeur la plus petite.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.PI

Syntaxe
```
Math.PI
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour le rapport de la circonférence d'un cercle à son diamètre, exprimée par pi, avec une valeur de 3,14159265358979.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.pow

Syntaxe

```
Math.pow(x, y);
```

Arguments

x Un nombre devant être élevé à la puissance.

y Un nombre spécifiant la puissance à laquelle l'argument *x* est élevé.

Description

Méthode ; calcule et renvoie *x* à la puissance *y*, x^y.

Lecteur

Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.random

Syntaxe

```
Math.random();
```

Arguments

Aucun.

Description

Méthode ; renvoie un nombre pseudo-aléatoire entre 0,0 et 1,0.

Lecteur

Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Voir aussi

« random » à la page 350

Math.round

Syntaxe

```
Math.round(x);
```

Arguments

x Tout nombre.

Description
Méthode ; arrondit la valeur de l'argument *x* à l'entier inférieur ou supérieur le plus proche et renvoie la valeur.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.sin

Syntaxe
```
Math.sin(x);
```

Arguments
x Un angle mesuré en radians.

Description
Méthode ; calcule et renvoie le sinus de l'angle spécifié, en radians. Utilisez les informations indiquées dans l'introduction de l'objet Math pour calculer un radian.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Voir aussi
« Math (objet) » à la page 296

Math.sqrt

Syntaxe
```
Math.sqrt(x);
```

Arguments
x Tout nombre, ou expression, supérieur ou égal à 0.

Description
Méthode ; calcule et renvoie la racine carrée du nombre spécifié.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.SQRT1_2

Syntaxe
```
Math.SQRT1_2
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour la réciproque de la racine carrée de un-demi (1/2), avec une valeur approximative de 0,707106781186.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.SQRT2

Syntaxe
```
Math.SQRT2
```

Arguments
Aucun.

Description
Constante ; une constante mathématique pour la racine carrée de 2, exprimée sous la forme avec une valeur approximative de 1,414213562373.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

Math.tan

Syntaxe
```
Math.tan(x);
```

Arguments
x Un angle mesuré en radians.

Description
Méthode ; calcule et renvoie la tangente de l'angle spécifié. Utilisez les informations indiquées dans l'introduction de l'objet Math pour calculer un radian.

Lecteur
Flash 5 et versions suivantes. Dans le Flash 4 Player, les méthodes et propriétés de l'objet Math sont émulées avec des approximations et peuvent ne pas être aussi précises que les fonctions math non émulées supportées par le Flash 5 Player.

maxscroll

Syntaxe
```
variable_name.maxscroll = x
```

Arguments
variable_name Le nom d'une variable associée avec un champ de texte.

x Le nombre de lignes maximum autorisé pour la propriété `scroll`, basé sur la hauteur du champ de texte. Il s'agit d'une valeur en lecture seule définie par Flash.

Description
Propriété ; une propriété en lecture seule qui fonctionne avec la propriété `scroll` pour contrôler l'affichage des informations dans un champ de texte. Cette propriété peut être extraite mais pas modifiée.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« scroll » à la page 353

mbchr

Syntaxe
```
mbchr(number);
```

Arguments
number Le nombre à convertir en caractère multioctet.

Description
Fonction de chaîne ; convertit un nombre en code ASCII en caractère multioctet.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5 ; l'utilisation de la méthode `String.fromCharCode` est encouragée.

Voir aussi
« String.fromCharCode » à la page 374

mblength

Syntaxe
`mblength(string);`

Arguments
string Une chaîne.

Description
Fonction de chaîne ; renvoie la longueur de la chaîne de caractères multioctets.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5 ; l'utilisation des méthodes et de l'objet Chaîne est recommandée.

mbord

Syntaxe
`mbord(character);`

Arguments
character Le caractère à convertir en nombre multioctet.

Description
Fonction de chaîne ; convertit le caractère spécifié en nombre multioctet.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5 ; l'utilisation de la méthode `String.charCodeAt` est recommandée.

Voir aussi
« String.fromCharCode » à la page 374

mbsubstring

Syntaxe
`mbsubstring(value, index, count);`

Arguments
value La chaîne multioctet de laquelle extraire une nouvelle chaîne multioctet.

index Le nombre du premier caractère à extraire.

count Le nombre de caractères à inclure dans la chaîne extraite, excepté le caractère index.

Description
Fonction de chaîne ; extrait une nouvelle chaîne de caractères multioctets à partir d'une chaîne de caractères multioctets.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5 ; l'utilisation de la méthode `string.substr` est recommandée.

Voir aussi
« String.substr » à la page 377

Mouse (objet)

Utilisez les méthodes de l'objet Mouse pour masquer et afficher le pointeur dans l'animation. Le pointeur est visible par défaut, mais vous pouvez masquer le pointeur et implémenter un pointeur personnalisé que vous créez en utilisant un clip.

Tableau des méthodes de l'objet Mouse

Méthode	Description
`hide`	Masque le pointeur dans l'animation.
`show`	Affiche le pointeur dans l'animation.

Mouse.hide

Syntaxe
```
Mouse.hide();
```

Arguments
Aucun.

Description
Méthode ; masque le pointeur dans une animation. Le pointeur est visible par défaut.

Lecteur
Flash 5 et versions suivantes.

Exemple

Le code suivant, associé à un clip du scénario principal, masque le pointeur standard et définit les positions *x* et *y* de l'occurrence de clip `customCursor` sur les positions *x* et *y* de la souris dans le scénario principal :

```
onClipEvent(enterFrame) {
   Mouse.hide();
   customCursorMC_x = _root._xmouse;
   customCursorMC_y = _root._ymouse;
}
```

Voir aussi

« _xmouse » à la page 418
« _ymouse » à la page 420
« Mouse.show » à la page 312w

Mouse.show

Syntaxe

```
Mouse.show();
```

Arguments

Aucun.

Description

Méthode ; rend le pointeur visible dans une animation. Le pointeur est visible par défaut.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« _xmouse » à la page 418
« _ymouse » à la page 420
« Mouse.show » à la page 312

MovieClip (objet)

Les méthodes de l'objet MovieClip fournissent la même fonctionnalité que les actions standard ciblant les clips. Il existe des méthodes supplémentaires qui fournissent une fonctionnalité indisponible dans les actions standard répertoriées dans la catégorie Actions du panneau Actions. Il n'est pas nécessaire d'utiliser une méthode constructeur pour appeler les méthodes de l'objet MovieClip ; à la place, vous faites référence aux occurrences de clips par leurs noms, en utilisant la syntaxe suivante :

```
anyMovieClip.play();
anyMovieClip.gotoAndPlay(3);
```

Tableau des méthodes de l'objet MovieClip

Méthode	Description
`attachMovie`	Associe une animation dans la bibliothèque.
`duplicateMovieClip`	Duplique le clip spécifié.
`getBounds`	Renvoie les coordonnées *x* et *y* minimum et maximum d'une animation dans un espace coordonné spécifié.
`getBytesLoaded`	Renvoie le nombre d'octets chargés pour le clip spécifié.
`getBytesTotal`	Renvoie la taille du clip en octets.
`getURL`	Extrait un document d'une URL.
`globalToLocal`	Convertit les coordonnées de scène de l'objet point en coordonnées locales du clip spécifié.
`gotoAndPlay`	Envoie la tête de lecture dans une image spécifique du clip et lit l'animation.
`gotoAndStop`	Envoie la tête de lecture dans une image spécifique du clip et stoppe l'animation.
`hitTest`	Renvoie `true` si le cadre de délimitation du clip spécifié croise le cadre de délimitation du clip cible.
`loadMovie`	Charge l'animation spécifiée dans le clip.
`loadVariables`	Charge les variables depuis une URL ou un autre emplacement dans le clip.
`localToGlobal`	Convertit les coordonnées locales de clip de l'objet point en coordonnées globales de scène.
`nextFrame`	Envoie la tête de lecture dans l'image suivante du clip.
`play`	Exécute le clip spécifié.
`prevFrame`	Envoie la tête de lecture dans l'image précédente du clip.
`removeMovieclip`	Supprime le clip du scénario s'il a été créé avec une action ou une méthode duplicateMovieClip ou avec la méthode attachMovie.
`startDrag`	Spécifie un clip comme glissable et commence à faire glisser le clip.
`stop`	Arrête la lecture de l'animation courante.

Méthode	Description
`stopDrag`	Arrête le glissement d'un clip qui est déplacé.
`swapDepths`	Permute le niveau de profondeur d'une animation spécifiée avec l'animation située à un niveau de profondeur spécifique.
`unloadMovie`	Supprime une animation chargée avec `loadMovie`.

MovieClip.attachMovie

Syntaxe

```
anyMovieClip.attachMovie(idName, newname, depth);
```

Arguments

idName Le nom de l'animation à associer dans la bibliothèque. Il s'agit du nom entré dans le champ Identifiant de la boîte de dialogue Propriétés de liaison de symbole.

newname Un nom d'occurrence unique pour le clip à associer.

depth Un entier spécifiant le niveau de profondeur où est placée l'animation.

Description

Méthode ; crée une nouvelle occurrence d'une animation dans la bibliothèque et l'associe à l'animation spécifiée par *anyMovieClip*. Utilisez les actions ou méthodes `removeMovieClip` ou `unloadMovie` pour supprimer une animation associée avec `attachMovie`.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« removeMovieClip » à la page 350
« unloadMovie » à la page 385
« MovieClip.removeMovieClip » à la page 324
« MovieClip.unloadMovie » à la page 326

MovieClip.duplicateMovieClip

Syntaxe

```
anyMovieClip.duplicateMovieClip(newname, depth);
```

Arguments

newname Un identifiant unique pour le clip dupliqué.

depth Un nombre spécifiant le niveau de profondeur où l'animation spécifiée doit être placée.

Description
Méthode ; crée une occurrence du clip spécifié pendant la lecture de l'animation. Les clips dupliqués commencent toujours la lecture à partir de l'image 1, peu importe sur quelle image se trouve l'animation originale lorsque la méthode `duplicateMovieClip` est appelée. Les variables du clip parent ne sont pas copiées dans le clip dupliqué. Si le clip parent est effacé, le clip dupliqué l'est également. Les clips ajoutés avec `duplicateMovieClip` peuvent être effacés avec l'action ou la méthode `removeMovieClip`.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« removeMovieClip » à la page 350
« MovieClip.removeMovieClip » à la page 324

MovieClip.getBounds

Syntaxe
anyMovieClip.getBounds(*targetCoordinateSpace*);

Arguments
targetCoordinateSpace Le chemin cible du scénario dont vous voulez utiliser l'espace coordonné comme point de référence.

Description
Méthode ; Renvoie les valeurs maximum et minimum des coordonnées *x* et *y* de MovieClip, pour l'espace coordonné spécifié dans l'argument. L'objet renvoyé contiendra les propriétés `{xMin, xMax, yMin, yMax}`. Utilisez les méthodes `localToGlobal` et `globalToLocal` de l'objet MovieClip pour convertir, respectivement, les coordonnées locales du clip en coordonnées de scène ou les coordonnées de scène en coordonnées locales.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant utilise `getBounds` pour extraire le cadre de délimitation de l'occurrence `myMovieClip` dans l'espace coordonné de l'animation principale.

```
myMovieClip.getBounds(._root);
```

Voir aussi
« MovieClip.globalToLocal » à la page 317
« MovieClip.localToGlobal » à la page 322

MovieClip.getBytesLoaded

Syntaxe
anyMovieClip.getBytesLoaded();

Arguments
Aucun.

Description
Méthode ; renvoie le nombre d'octets chargés (émis) pour l'objet MovieClip spécifié. Étant donné que les clips internes se chargent automatiquement, le résultat renvoyé pour cette méthode et pour `MovieClip.getBytesTotal` sera le même si l'objet MovieClip spécifié fait référence à un clip interne. Cette méthode est prévue pour être utilisée avec des animations chargées. Vous pouvez comparer la valeur de `getBytesLoaded` avec la valeur de `getBytesTotal` pour déterminer quel pourcentage d'une animation externe a été chargé.

Lecteur
Flash 5 et versions suivantes.

MovieClip.getBytesTotal

Syntaxe
anyMovieClip.getBytesTotal();

Arguments
Aucun.

Description
Méthode ; renvoie la taille, en octets, de l'objet MovieClip spécifié. Pour les clips externes (l'animation racine ou un clip chargé dans une cible ou un niveau), la valeur renvoyée correspond à la taille du fichier SWF.

Lecteur
Flash 5 et versions suivantes.

MovieClip.getURL

Syntaxe

```
anyMovieClip.getURL(URL [,window, variables]]);
```

Arguments

URL L'URL d'où obtenir le document.

window Un argument optionnel spécifiant le nom, l'image ou l'expression spécifiant la fenêtre ou l'image HTML dans lequel le document se charge. Vous pouvez également utiliser un des noms cible réservés suivants : `_self` spécifie l'image courante dans la fenêtre courante, `_blank` spécifie une nouvelle fenêtre, `_parent` spécifie le parent de l'image courante, `_top` spécifie l'image de haut niveau dans la fenêtre courante.

variables Un argument optionnel spécifiant une méthode pour envoyer les variables associées à l'animation à charger. S'il n'y a pas de variables, passez cet argument ; sinon, spécifiez si les variables doivent être chargées avec une méthode `GET` ou `POST`. `GET` ajoute les variables à la fin de l'URL et est utilisée pour des petits nombres de variables. `POST` envoie les variables dans un en-tête HTTP séparé et est utilisée pour les longues chaînes de variables.

Description

Méthode ; charge un document depuis une URL spécifiée dans la fenêtre spécifiée. La méthode `getURL` peut également être utilisée pour transmettre des variables dans une autre application définie à l'URL en utilisant une méthode `GET` ou `POST`.

Lecteur

Flash 5 et versions suivantes.

MovieClip.globalToLocal

Syntaxe

```
anyMovieClip.globalToLocal(point);
```

Arguments

point Le nom ou l'identifiant d'un objet créé avec l'objet générique Objet spécifiant les coordonnées *x* et *y* comme propriétés.

Description

Méthode ; convertit les coordonnées de scène (globales) de l'objet *point* en coordonnées de clip (locales).

Lecteur

Flash 5 et versions suivantes.

Exemple
L'exemple suivant convertit les coordonnées *x* et *y* globales de l'objet *point* en coordonnées locales du clip.

```
onClipEvent(mouseMove) {
point = new object();
point.x = _root._xmouse;
point.y = _root._ymouse;
globalToLocal(point);
_root.out = _xmouse + " === " + _ymouse;
_root.out2 = point.x + " === " + point.y;
updateAfterEvent();
}
```

Voir aussi
« MovieClip.localToGlobal » à la page 322
« MovieClip.getBounds » à la page 315

MovieClip.gotoAndPlay

Syntaxe
anyMovieClip.gotoAndPlay(*frame*);

Arguments
frame Le numéro de l'image dans laquelle la tête de lecture sera envoyée.

Description
Méthode ; démarre la lecture de l'animation à partir de l'image spécifiée.

Lecteur
Flash 5 et versions suivantes.

MovieClip.gotoAndStop

Syntaxe
anyMovieClip.gotoAndStop(*frame*);

Arguments
frame Le numéro de l'image dans laquelle la tête de lecture sera envoyée.

Description
Méthode ; arrête la lecture de l'animation à l'image spécifiée.

Lecteur
Flash 5 et versions suivantes.

MovieClip.hitTest

Syntaxe

```
anyMovieClip.hitTest(x, y, shapeFlag);
anyMovieClip.hitTest(target);
```

Arguments

x La coordonnée *x* de la zone d'accès sur la scène.

y La coordonnée *y* de la zone d'accès sur la scène.

Les coordonnées *x* et *y* sont définies dans l'espace coordonné global.

target Le chemin cible de la zone d'accès pouvant croiser ou recouvrir l'occurrence spécifiée par *anyMovieClip*. L'argument *target* représente généralement un bouton ou un champ de saisie de texte.

shapeFlag Une valeur booléenne spécifiant s'il faut évaluer la forme entière de l'occurrence spécifiée (*true*), ou seulement le cadre de délimitation (*false*). Cet argument ne peut que spécifier si la zone d'accès est identifiée en utilisant les arguments de coordonnées *x* et *y*.

Description

Méthode ; évalue l'occurrence spécifiée par *anyMovieClip* pour voir si elle recouvre ou croise la zone d'accès identifiée par *target* ou les arguments de coordonnées *x* et *y*.

Emploi 1 : compare les coordonnées *x* et *y* avec la forme ou le cadre de délimitation de l'occurrence spécifiée, en fonction du paramètre *shapeFlag*. Si *shapeFlag* est défini sur `true`, seule la zone occupée actuellement par l'occurrence sur la scène est évaluée et si *x* et *y* recouvrent un point, une valeur `true` est renvoyée. Cela est utile pour déterminer si le clip se trouve dans une zone spécifiée d'accès ou sensible.

Emploi 2 : évalue les cadres de délimitation de *target* et de l'occurrence spécifiée et renvoie `true` s'ils se croisent ou se recouvrent en un point.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `hitTest` avec les propriétés `x_mouse` et `y_mouse` pour déterminer si la souris se trouve sur le cadre de délimitation de la cible.

```
if (hitTest( _root._xmouse, _root._ymouse, false));
```

L'exemple suivant utilise `hitTest` pour déterminer si le clip `ball` recouvre ou croise le clip `square`.

```
if(_root.ball, hittest(_root.square)){
trace("ball intersects square");
}
```

Voir aussi

« MovieClip.localToGlobal » à la page 322
« MovieClip.globalToLocal » à la page 317
« MovieClip.getBounds » à la page 315

MovieClip.loadMovie

Syntaxe

anyMovieClip.loadMovie(*url* [,*variables*]);

Arguments

url Une URL absolue ou relative pour le fichier SWF à charger. Un chemin relatif doit être relatif au SWF. L'URL doit se trouver dans le même sous-domaine que l'URL où se trouve l'animation courante. Pour une utilisation dans le Flash Player ou pour un test en mode test d'animation dans l'environnement de programmation de Flash, tous les fichiers SWF doivent être stockés dans le même dossier et les noms de fichiers ne doivent pas inclure de spécifications de dossier ni d'unité de disque.

variables Un argument optionnel spécifiant une méthode pour envoyer les variables associées à l'animation à charger. Cet argument doit être la chaîne « GET » ou « POST ». S'il n'y a pas de variables, passez cet argument ; sinon, spécifiez si les variables doivent être chargées avec une méthode `GET` ou `POST`. `GET` ajoute les variables à la fin de l'URL et est utilisée pour des petits nombres de variables. `POST` envoie les variables dans un en-tête HTTP séparé et est utilisée pour les longues chaînes de variables.

Description
Méthode ; lit des animations supplémentaires sans fermer le Flash Player. Normalement, le Flash Player affiche une seule animation Flash Player (fichier SWF) puis ferme. La méthode `loadMovie` vous permet d'afficher plusieurs animations en même temps ou de passer d'une animation à l'autre sans charger un autre document HTML.

Utilisez l'action `unloadMovie` pour supprimer les animations chargées avec l'action `loadMovie`.

Utilisez la méthode `loadVariables` pour garder l'animation active et mettre à jour les variables avec de nouvelles valeurs.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« MovieClip.loadVariables » à la page 321
« MovieClip.unloadMovie » à la page 326

MovieClip.loadVariables

Syntaxe
anyMovieClip.loadVariables(*url, variables*);

Arguments
url L'URL relative ou absolue pour le fichier externe. L'hôte de l'URL doit se trouver dans le même sous-domaine que le clip.

variables La méthode pour extraire les variables. `GET` ajoute les variables à la fin de l'URL et est utilisée pour des petits nombres de variables. `POST` envoie les variables dans un en-tête HTTP séparé et est utilisée pour les longues chaînes de variables.

Description
Méthode ; lit les données depuis un fichier externe et définit les valeurs pour les variables dans une animation ou dans un clip. Le fichier externe peut être un fichier texte généré par un script CGI, Active Server Pages (ASP) ou PHP, et peut contenir n'importe quel nombre de variables.

Cette méthode peut également être utilisée pour mettre à jour des variables dans l'animation active avec de nouvelles valeurs.

Cette méthode nécessite que le texte de l'URL soit au format standard MIME : *application/x-www-urlformencoded* (format script CGI).

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« MovieClip.loadMovie » à la page 320

MovieClip.localToGlobal

Syntaxe

anyMovieClip.localToGlobal(*point*);

Arguments

point Le nom ou l'identifiant d'un objet créé avec l'objet Objet, spécifiant les coordonnées *x* et *y* comme propriétés.

Description

Méthode ; convertit les coordonnées du clip (locales) de l'objet *point* en coordonnées de scène (globales).

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant convertit les coordonnées *x* et *y* de l'objet *point*, à partir des coordonnées du clip (locales) en coordonnées de scène (globales). Les coordonnées locales *x* et *y* sont spécifiées en utilisant xmouse et ymouse pour extraire les coordonnées *x* et *y* de la position de la souris.

```
onClipEvent(mouseMove) {
point = new object();
point.x = _xmouse;
point.y = _ymouse;
_root.out3 = point.x + " === " + point.y;
_root.out = _root._xmouse + " === " + _root._ymouse;
localToGlobal(point);
_root.out2 = point.x + " === " + point.y;
updateAfterEvent();
}
```

Voir aussi

« MovieClip.globalToLocal » à la page 317

MovieClip.nextFrame

Syntaxe
```
anyMovieClip.nextFrame();
```

Arguments
Aucun.

Description
Méthode ; envoie la tête de lecture dans l'image suivante du clip.

Lecteur
Flash 5 et versions suivantes.

MovieClip.play

Syntaxe
```
anyMovieClip.play();
```

Arguments
Aucun.

Description
Méthode ; lit le clip.

Lecteur
Flash 5 et versions suivantes.

MovieClip.prevFrame

Syntaxe
```
anyMovieClip.prevFrame();
```

Arguments
Aucun.

Description
Méthode ; envoie la tête de lecture dans l'image précédente et l'arrête.

Lecteur
Flash 5 et versions suivantes.

MovieClip.removeMovieClip

Syntaxe
anyMovieClip.removeMovieClip();

Arguments
Aucun.

Description
Méthode ; supprime une occurrence de clip créée avec l'action duplicateMovieclip ou avec les méthodes duplicateMovieClip ou attachMovie de l'objet MovieClip.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« MovieClip.loadMovie » à la page 320
« MovieClip.attachMovie » à la page 314

MovieClip.startDrag

Syntaxe
anyMovieClip.startDrag([*lock*, *left*, *right*, *top*, *bottom*]);

Arguments
lock Une valeur booléenne spécifiant si le clip glissable est verrouillé au centre de la position de la souris (*true*), ou verrouillé sur le point où l'utilisateur a cliqué la première fois dans l'animation (*false*). Cet argument est optionnel.

left, *top*, *right*, *bottom* Valeurs relatives aux coordonnées du parent du clip qui spécifient un rectangle contraint pour le clip. Ces arguments sont optionnels.

Description
Méthode ; permet à l'utilisateur de faire glisser le clip spécifié. L'animation reste glissable jusqu'à ce qu'elle soit explicitement arrêtée par l'appel de la méthode stopDrag ou jusqu'à ce qu'un autre clip devienne glissable. On ne peut faire glisser qu'un seul clip à la fois.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« MovieClip.stopDrag » à la page 325
« _droptarget » à la page 262

MovieClip.stop

Syntaxe
anyMovieClip.stop();

Arguments
Aucun.

Description
Méthode ; arrête la lecture du clip courant.

Lecteur
Flash 5 et versions suivantes.

MovieClip.stopDrag

Syntaxe
anyMovieClip.stopDrag();

Arguments
Aucun.

Description
Méthode ; arrête une action de glissement implémentée avec la méthode `startDrag`. Une animation reste glissable jusqu'à ce qu'une méthode `stopDrag` soit ajoutée ou jusqu'à ce qu'une autre animation devienne glissable. On ne peut faire glisser qu'un seul clip à la fois.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« _droptarget » à la page 262
« MovieClip.startDrag » à la page 324

MovieClip.swapDepths

Syntaxe
anyMovieClip.swapDepths(*depth*);

anyMovieClip.swapDepths(*target*);

Arguments
target L'occurrence du clip dont la profondeur est permutée avec l'occurrence spécifiée par *anyMovieClip*. Les deux occurrences doivent avoir le même clip parent.

depth Un nombre spécifiant le niveau de profondeur où *anyMovieClip* doit être placé.

Description
Méthode ; permute l'ordre d'empilage (niveau de profondeur), ou *z*, de l'occurrence spécifiée avec l'animation spécifiée par l'argument *target*, ou avec l'animation occupant actuellement le niveau *depth* spécifié dans l'argument. Les deux animations doivent avoir le même clip parent. Permuter le niveau de profondeur des clips a pour effet de déplacer une animation devant ou derrière l'autre. Si une animation est en train d'interpoler lorsque cette méthode est appelée, l'interpolation est arrêtée.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« _level » à la page 292

MovieClip.unloadMovie

Syntaxe
```
anyMovieClip.unloadMovie();
```

Arguments
Aucun.

Description
Méthode ; supprime un clip chargé avec les méthodes `loadMovie` ou `attachMovie`.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« MovieClip.loadMovie » à la page 320
« MovieClip.attachMovie » à la page 314

_name

Syntaxe
```
instancename._name
instancename._name = value;
```

Arguments
instancename Un nom d'occurrence de clip pour lequel la propriété `_name` doit être définie ou extraite.

value Une chaîne qui spécifie un nouveau nom d'occurrence.

Description
Propriété ; spécifie le nom d'occurrence du clip.

Lecteur
Flash 4 et versions suivantes.

NaN

Syntaxe
```
NaN
```

Arguments
Aucun.

Description
Variable ; une variable prédéfinie avec la valeur IEEE-754 pour NaN (Not a Number).

Lecteur
Flash 5 et versions suivantes.

ne (inégalité–propre aux chaînes)

Syntaxe
expression1 `ne` *expression2*

Arguments
expression1, expression2 Nombres, chaînes ou variables.

Description
Opérateur (comparaison) ; compare *expression1* avec *expression2* et renvoie `true` si *expression1* n'est pas égale à *expression2* ; sinon, renvoie `false`.

Lecteur
Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur `!=` (inégalité) est recommandée.

Voir aussi
« != (inégalité) » à la page 191

new

Syntaxe

```
new constructor();
```

Arguments

constructor Une fonction suivie par tout argument optionnel entre parenthèses. La fonction est généralement le nom du type de l'objet (par exemple, Chaîne, Math, Nombre, Objet) à construire.

Description

Opérateur ; crée un nouvel objet (anonyme au début), appelle la fonction identifiée par l'argument *constructor* , transmet tous les arguments optionnels entre parenthèses et transmet l'objet nouvellement créé comme valeur du mot-clé `this`. La fonction constructeur peut ensuite utiliser `this` pour instancier le nouvel objet.

La propriété _prototype_ de l'objet de la fonction constructeur est copiée dans la propriété _proto_ du nouvel objet. En conséquence, le nouvel objet supporte toutes les méthodes et propriétés spécifiées dans l'objet prototype de la fonction constructeur.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant crée les objets `book1` et `book2` avec l'opérateur new.

```
function Book(name, price)
{
  this.name = name;
  this.price = price;
}
book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```

Voir aussi

« [] (opérateur accès tableau) » à la page 203
« {} (initialisateur d'objet) » à la page 205
La section méthode constructeur à l'entrée d'un objet.

newline

Syntaxe
```
newline;
```

Arguments
Aucun.

Description
Constante ; insère un caractère de retour chariot ({) insérant une ligne vide dans le code ActionScript. Utilisez `newline` pour faire de la place aux informations extraites par une fonction ou une action dans votre code.

Lecteur
Flash 4 et versions suivantes.

nextFrame

Syntaxe
```
nextFrame();
```

Arguments
Aucun.

Description
Action ; envoie la tête de lecture dans l'image suivante et l'arrête.

Lecteur
Flash 2 et versions suivantes.

Exemple
Lorsqu'un utilisateur clique sur un bouton auquel l'action `nextFrame` est affectée, la tête de lecture est envoyée sur l'image suivante.

```
on (release) {
    nextFrame(5);
}
```

nextScene

Syntaxe
```
nextScene();
```

Arguments
Aucun.

Description
Action ; envoie la tête de lecture sur l'image 1 de la scène suivante et l'arrête.

Lecteur
Flash 2 et versions suivantes.

Exemple
Cette action est affectée à un bouton qui, lorsqu'il est pressé puis relâché, envoie la tête de lecture dans l'image 1 de la scène suivante.

```
on(release) {
    nextScene();
}
```

not

Syntaxe
`not expression`

Arguments
`expression` Toute variable ou expression qui se convertit en valeur booléenne.

Description
Opérateur ; effectue une opération logique NOT dans le Flash 4 Player.

Lecteur
Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 ; l'utilisation du nouvel opérateur `!` (NOT logique) est recommandée.

Voir aussi
« ! (NOT logique) » à la page 191

null

Syntaxe
`null`

Arguments
Aucun.

Description
Mot-clé ; une valeur spéciale qui peut être affectée aux variables ou renvoyée par une fonction si aucune donnée n'a été fournie. Vous pouvez utiliser `null` pour représenter les valeurs manquantes ou n'ayant pas de type de données défini.

Lecteur
Flash 5 et versions suivantes.

Exemple
Dans un contexte numérique, `null` évalue 0. Les tests d'égalité peuvent être effectués avec `null`. Dans cette instruction, un TreeNode binaire ne possède pas d'enfant left, donc le champ pour cet enfant left peut être défini sur `null`.

```
if (tree.left == null) {
  tree.left = new TreeNode();
}
```

Number (fonction)

Syntaxe
`Number(`*`expression`*`);`

Arguments
`expression` La chaîne, le booléen ou une autre expression à convertir en nombre.

Description
Fonction ; convertit l'argument *x* en nombre et renvoie une valeur selon les règles suivantes :

Si *x* est un nombre, la valeur renvoyée est *x*.

Si *x* est un booléen, la valeur renvoyée est 1 si *x* est `true`, 0 si *x* est `false`.

Si *x* est une chaîne, la fonction tente d'analyser *x* en nombre décimal avec un exposant à droite optionnel, c'est-à-dire, 1,57505e-3.

Si *x* est indéfini, la valeur renvoyée est 0.

Cette fonction est utilisée pour convertir les fichiers Flash 4 contenant des opérateurs désapprouvés importés dans l'environnement de programmation de Flash 5. Voir l'opérateur `&` pour plus d'informations.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« Nombre (objet) » à la page 332

Nombre (objet)

L'objet Nombre est un simple objet enveloppe pour le type de données number, ce qui signifie que vous pouvez manipuler des valeurs numériques primitives en utilisant les méthodes et les propriétés associées à l'objet Nombre. La fonctionnalité fournie par cet objet est identique à celle de l'objet JavaScript Nombre.

Vous devez utiliser le constructeur `Number` lorsque vous appelez les méthodes de l'objet Nombre, mais il n'est pas nécessaire d'utiliser le constructeur lorsque vous appelez les propriétés de l'objet Nombre. Les exemples suivants spécifient la syntaxe à employer pour appeler les méthodes et les propriétés de l'objet Nombre :

Cet exemple appelle la méthode `toString` de l'objet Nombre :

```
myNumber = new Number(1234);
myNumber.toString();
```

Renvoie une chaîne contenant la représentation binaire du nombre 1234.

Cet exemple appelle la propriété (également nommée constante) `MIN_VALUE` de l'objet Nombre :

```
smallest = Number.MIN_VALUE
```

Tableau des méthodes de l'objet Nombre

Méthode	Description
`toString`	Renvoie la représentation chaîne d'un objet Nombre.
`valueOf`	Renvoie la valeur primitive d'un objet Nombre.

Tableau des propriétés de l'objet Nombre

Propriété	Description
`MAX_VALUE`	Constante représentant le plus grand nombre représentable (IEEE-754 double précision). Ce nombre est approximativement 1,7976931348623158e+308.
`MIN_VALUE`	Constante représentant le plus petit nombre représentable (IEEE-754 double précision). Ce nombre est approximativement 5e-324.
`NaN`	Constante représentant la valeur de Not a Number (NaN).
`NEGATIVE_INFINITY`	Constante représentant la valeur pour l'infini négatif.
`POSITIVE_INFINITY`	Constante représentant la valeur pour l'infini positif. Cette valeur est la même que la variable globale Infinity.

Constructeur pour l'objet Nombre

Syntaxe

```
myNumber = new Number(value);
```

Arguments

value La valeur numérique de l'objet Nombre en création, ou une valeur à convertir en nombre.

Description

Constructeur ; crée un nouvel objet Nombre. Vous devez utiliser le constructeur `Number` lorsque vous utilisez les méthodes `toString` et `valueOf` de l'objet Nombre. Il n'est pas nécessaire d'utiliser le constructeur lorsque vous utilisez les propriétés de l'objet Nombre. Le constructeur `new Number` est d'abord utilisé comme support. Une occurrence de l'objet Nombre n'est pas identique à la fonction `Number` qui convertit un argument en valeur primitive.

Lecteur

Flash 5 et versions suivantes.

Exemple

Le code suivant construit les objets new Number.

```
n1 = new Number(3.4);
n2 = new Number(-10);
```

Voir aussi

« Number (fonction) » à la page 331

Number.MAX_VALUE

Syntaxe

```
Number.MAX_VALUE
```

Arguments

Aucun.

Description

Propriété ; le nombre représentable le plus grand (IEEE-754 double précision). Ce nombre est approximativement 1,79E+308.

Lecteur

Flash 5 et versions suivantes.

Number.MIN_VALUE

Syntaxe
`Number.MIN_VALUE`

Arguments
Aucun.

Description
Propriété ; le nombre représentable le plus petit (IEEE-754 double précision). Ce nombre est approximativement 5e-324.

Lecteur
Flash 5 et versions suivantes.

Number.NaN

Syntaxe
`Number.NaN`

Arguments
Aucun.

Description
Propriété ; la valeur IEEE-754 représentant Not A Number (`NaN`).

Lecteur
Flash 5 et versions suivantes.

Number.NEGATIVE_INFINITY

Syntaxe
`Number.NEGATIVE_INFINITY`

Arguments
Aucun.

Description
Propriété ; renvoie la valeur IEEE-754 représentant l'infini négatif. Cette valeur est identique à la variable globale `Infinity`.

L'infini négatif est une valeur numérique spéciale qui est renvoyée lorsqu'une opération mathématique ou une fonction renvoient une valeur négative trop grande pour être représentée.

Lecteur
Flash 5 et versions suivantes.

Number.POSITIVE_INFINITY

Syntaxe

```
Number.POSITIVE_INFINITY
```

Arguments

Aucun.

Description

Propriété ; renvoie la valeur IEEE-754 représentant l'infini positif. Cette valeur est identique à la variable globale `Infinity`.

L'infini positif est une valeur numérique spéciale qui est renvoyée lorsqu'une opération mathématique ou une fonction renvoient une valeur trop grande pour être représentée.

Lecteur

Flash 5 et versions suivantes.

Number.toString

Syntaxe

```
myNumber.toString(radix);
```

Arguments

`radix` Spécifie la base numérique (entre 2 et 36) à utiliser pour la conversion d'un nombre en chaîne. Si vous ne spécifiez pas l'argument *`radix`*, la valeur par défaut est 10.

Description

Méthode ; renvoie la représentation chaîne de l'objet Nombre spécifié (*`myNumber`*).

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise la méthode `Number.toString`, en spécifiant 2 pour l'argument *`radix`* :

```
myNumber = new Number (1000);
(1000).toString(2);
```

Renvoie une chaîne contenant la représentation binaire du nombre 1000.

Number.valueOf

Syntaxe
```
myNumber.valueOf();
```

Arguments
Aucun.

Description
Méthode ; renvoie le type de valeur primitive de l'objet Nombre spécifié et convertit l'objet enveloppe Nombre en cette valeur primitive.

Lecteur
Flash 5 et versions suivantes.

Objet (objet)

L'objet générique Objet est à la base de la hiérarchie ActionScript. La fonctionnalité de l'objet générique Objet est un petit sous-ensemble de celle fournit par l'objet JavaScript Objet.

L'objet générique Objet nécessite le Flash 5 Player.

Tableau des méthodes de l'objet Objet

Méthode	Description
`toString`	Convertit l'objet spécifié en chaîne et le renvoie.
`valueOf`	Renvoie la valeur primitive d'un objet Objet.

Constructeur pour l'objet Objet

Syntaxe
```
new Object();
new Object(value);
```

Arguments
value Un nombre, un booléen ou une chaîne à convertir en objet. Cet argument est optionnel. Si vous ne spécifiez pas *value*, le constructeur crée un nouvel objet sans propriétés définies.

Description
Constructeur ; crée un nouvel objet Objet.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« Sound.setTransform » à la page 363
« Color.setTransform » à la page 239

Object.toString

Syntaxe
```
myObject.toString();
```

Arguments
Aucun.

Description
Méthode ; convertit l'objet spécifié en chaîne et le renvoie.

Lecteur
Flash 5 et versions suivantes.

Object.valueOf

Syntaxe
```
myObject.valueOf();
```

Arguments
Aucun.

Description
Méthode ; renvoie la valeur primitive de l'objet spécifié. Si l'objet ne possède pas de valeur primitive, il est lui-même renvoyé.

Lecteur
Flash 5 et versions suivantes.

onClipEvent

Syntaxe

```
onClipEvent(movieEvent);{
...
}
```

Arguments

Un *movieEvent* est un événement déclencheur qui exécute les actions affectées à une occurrence de clip. N'importe laquelle des valeurs suivantes peut être spécifiée pour l'argument *movieEvent*.

- `load` L'action est initiée dès que le clip est instancié et apparaît dans le scénario.
- `unload` L'action est initiée dans la première image après que le clip soit supprimé du scénario. Les actions associées avec l'événement de clip `Unload` sont effectuées avant que des actions ne soient associées à l'image affectée.
- `enterFrame` L'action est initiée à chaque lecture d'images, comme les actions associées à un clip. Les actions associées avec l'événement de clip `OnEnterFrame` sont effectuées après les actions associées aux images affectées.
- `mouseMove` L'action est initiée chaque fois que la souris est bougée. Utilisez les propriétés `_xmouse` et `_ymouse` pour déterminer la position courante de la souris.
- `mouseDown` L'action est initiée quand le bouton gauche de la souris est pressé.
- `mouseUp` L'action est initiée quand le bouton gauche de la souris est relâché.
- `keyDown` L'action est initiée quand une touche est pressée. Utilisez la méthode `Key.getCode` pour extraire les informations concernant la dernière touche enfoncée.
- `keyUp` L'action est initiée quand une touche est relâchée. Utilisez la méthode `Key.getCode` pour extraire les informations concernant la dernière touche enfoncée.
- `data` L'action est initiée lorsque des données sont reçues dans une action `loadVariables` ou `loadMovie`. Lorsqu'il est spécifié avec une action `loadVariables`, l'événement `data` ne survient qu'une seule fois, quand la dernière variable est chargée. Lorsqu'il est spécifié avec une action `loadMovie`, l'événement `data` est répété plusieurs fois, au fur et à mesure que les sections de données sont extraites.

Description

Gestionnaire ; déclenche des actions définies pour une occurrence de clip spécifique.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'instruction suivante inclut le script d'un fichier externe lorsque l'occurrence de clip est chargée et apparaît la première fois dans le scénario.

```
onClipEvent(load) {
    #include "myScript.as"
}
```

L'exemple suivant utilise `onClipEvent` avec l'événement de clip `keyDown`. L'événement de clip `keyDown` est généralement utilisé en conjonction avec une ou plusieurs méthodes et propriétés associées à l'objet Touche. Dans le script suivant, `key.getCode` est utilisé pour trouver quelle touche l'utilisateur a pressée, la valeur renvoyée est associée avec les propriétés `RIGHT` ou `LEFT` de l'objet Touche et l'animation est dirigée en fonction.

```
onClipEvent(keyDown) {
    if (Key.getCode() == Key.RIGHT) {
      } _parent.nextFrame();
    else if (Key.getCode() == Key.LEFT){
       _parent.prevFrame();
}
```

L'exemple suivant utilise `onClipEvent` avec l'événement de clip `mouseMove`. Les propriétés `xmouse` et `ymouse` cherchent la position de la souris.

```
onClipEvent(mouseMove) {
stageX=_root.xmouse;
stageY=_root.ymouse;
}
```

Voir aussi

« on(mouseEvent) » à la page 340
« Key (objet) » à la page 282
« _xmouse » à la page 418
« _ymouse » à la page 420

on(mouseEvent)

Syntaxe

```
on(mouseEvent) {
statement;
}
```

Arguments

statement Les instructions à exécuter lorsque mouseEvent survient.

Une action *mouseEvent* peut avoir un des arguments suivants :

- `press` Le bouton de souris est pressé alors que le pointeur se trouve sur le bouton.
- `release` Le bouton de souris est relâché alors que le pointeur se trouve sur le bouton.
- `releaseOutside` Le bouton de souris est relâché alors que le pointeur se trouve en dehors du bouton.
- `rollOver` Le pointeur de la souris passe sur le bouton.
- `rollOut` Le pointeur passe en dehors de la zone du bouton.
- `dragOver` Alors que le pointeur se trouvait sur le bouton, le bouton de la souris a été pressé tout en se déplaçant en dehors du bouton puis il est revenu sur le bouton.
- `dragOut` Alors que le pointeur se trouvait sur le bouton, le bouton de la souris a été pressé puis déplacé en dehors de la zone du bouton.
- `keyPress (« key »)` La *key* spécifiée est enfoncée. La partie *key* de l'argument est spécifiée en utilisant un des codes-clés répertoriés dans l'Annexe B, « Touches de clavier et valeurs des codes-clés », ou une des constantes de Key répertoriées dans « Tableau des propriétés de l'objet Key » à la page 282.

Description

Gestionnaire ; spécifie l'événement de souris ou la frappe d'une touche qui déclenchent une action.

Lecteur

Flash 2 et versions suivantes.

Exemple

Dans le script suivant, l'action `startDrag` s'exécute lorsque la souris est pressée et le script conditionnel est exécuté lorsque la souris est relâchée ainsi que l'objet :

```
on(press) {
    startDrag("rabbit");
}
on(release) {
    if(getproperty("", _droptarget) == target) {
        setProperty ("rabbit", _x, _root.rabbit_x);
        setProperty ("rabbit", _y, _root.rabbit_y);
    } else {
_root.rabbit_x = getProperty("rabbit", _x);
_root.rabbit_y = getProperty("rabbit", _y);
_root.target = "pasture";
    }
    trace(_root.rabbit_y);
    trace(_root.rabbit_x);
    stopDrag();
}
```

Voir aussi

« Key (objet) » à la page 282
« onClipEvent » à la page 338

or

Syntaxe

condition1 or *condition2*

Arguments

condition1, condition2 Une expression qui évalue `true` ou `false`.

Description

Opérateur ; évalue *condition1* et *condition2* et si une des expressions est `true`, alors l'expression entière est `true`.

Lecteur

Flash 4 et versions suivantes. Cet opérateur a été désapprouvé dans Flash 5 et les utilisateurs sont encouragés à utiliser le nouvel opérateur `||` .

Voir aussi

« || (OR) » à la page 207

ord

Syntaxe
```
ord(character);
```

Arguments
character Le caractère à convertir en nombre de code ASCII.

Description
Fonction de chaîne ; convertit les caractères en nombres de code ASCII.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5 et il est recommandé d'utiliser les méthodes et propriétés de l'objet `String` à la place.

Voir aussi
« Chaîne (objet) » à la page 371

_parent

Syntaxe
```
_parent.property = x
_parent._parent.property = x
```

Arguments
property La propriété spécifiée pour le clip parent courant.

x La valeur définie pour la propriété. Cet argument est optionnel et il n'est pas nécessaire de le définir en fonction de la propriété.

Description
Propriété ; spécifie ou renvoie une référence au clip contenant le clip courant. Le clip courant est le clip contenant le script en cours d'exécution. Utilisez `_parent` pour spécifier un chemin relatif.

Lecteur
Flash 4 et versions suivantes.

Exemple
Dans l'exemple suivant, le clip `desk` est un enfant du clip `classroom`. Lorsque le script ci-dessous sera exécuté dans le clip `desk`, la tête de lecture ira dans l'image 10 du scénario du clip `classroom`.

```
_parent.gotoAndStop(10);
```

Voir aussi
« _root » à la page 351
« targetPath » à la page 379

parseFloat

Syntaxe
`parseFloat(string);`

Arguments
string La chaîne à analyser et à convertir en nombre à virgule flottante.

Description
Fonction ; convertit une chaîne en nombre à virgule flottante. La fonction analyse et renvoie les nombres de la chaîne jusqu'à ce que le programme d'analyse atteigne un caractère qui n'appartient pas au nombre initial. Si la chaîne ne commence pas par un nombre analysable, `parseFloat` renvoie `NaN` ou 0. Les espaces précédant les entiers valides sont ignorés tout comme les caractères non numériques à droite.

Lecteur
Flash 5 et versions suivantes.

Exemple
Les exemples suivants utilisent `parseFloat` pour évaluer différents types de nombres :

`parseFloat("-2")` renvoie `-2`

`parseFloat("2.5")` renvoie `2.5`

`parseFloat("3.5e6")` renvoie `3.5e6`, ou `3500000`

`parseFloat("foobar")` renvoie `NaN`

parseInt

Syntaxe
`parseInt(expression, radix);`

Arguments
expression La chaîne, le nombre à virgule flottante ou une autre expression à analyser et à convertir en entier

radix Un entier représentant le radical (base) du nombre à analyser. Les valeurs légales se situent entre 2 et 36. Cet argument est optionnel.

Description
Fonction ; convertit une chaîne en entier. Si la chaîne spécifiée dans les arguments ne peut pas être convertie en nombre, la fonction renvoie `NaN` ou 0. Les entiers commençant par 0 ou spécifiant une base de 8 sont interprétés comme des nombres octaux. Les entiers commençant par 0x sont interprétés comme des nombres hexadécimaux. Les espaces précédant les entiers valides sont ignorés tout comme les caractères non numériques à droite.

Lecteur
Flash 5 et versions suivantes.

Exemple
Les exemples suivants utilisent `parseInt` pour évaluer différents types de nombres.

`parseInt("3.5")` renvoie `3.5`

`parseInt("bar")` renvoie `NaN`

`parseInt("4foo")` renvoie `4`

Exemple
Conversion hexadécimale :

`parseInt("0x3F8")` renvoie `1016`

`parseInt("3E8", 16)` renvoie `1000`

Conversion binaire :

`parseInt("1010", 2)` renvoie `10` (la représentation décimale du binaire 1010)

Analyse d'un nombre octal (dans ce cas, le nombre octal est identifié par la base 8) :

`parseInt("777", 8)` renvoie `511` (la représentation décimale de l'octal 777)

play

Syntaxe
```
play();
```

Arguments
Aucun.

Description
Action ; fait avancer la tête de lecture dans le scénario.

Lecteur
Flash 2 et versions suivantes.

Exemple

Le code suivant utilise une instruction `if` pour vérifier la valeur d'un nom entré par l'utilisateur. Si l'utilisateur entre `Steve`, l'action `play` est appelée et la tête de lecture avance dans le scénario. Si l'utilisateur entre autre chose que `Steve`, l'animation n'est pas lue et un champ de texte contenant le nom de variable `alert` est affiché.

```
stop();
if (name = "Steve") {
    play();
} else {
    alert = "You are not Steve!";
}
```

prevFrame

Syntaxe

```
prevFrame();
```

Arguments

Aucun.

Description

Action ; envoie la tête de lecture dans l'image précédente et l'arrête.

Lecteur

Flash 2 et versions suivantes.

Exemple

Lorsque l'utilisateur clique sur un bouton auquel l'action `prevFrame` est affectée, la tête de lecture est envoyée dans l'image précédente.

```
on(release) {
    prevFrame(5);
}
```

Voir aussi

« MovieClip.prevFrame » à la page 323

prevScene

Syntaxe
```
prevScene();
```

Arguments
Aucun.

Description
Action ; envoie la tête de lecture dans l'image 1 de la scène précédente et l'arrête.

Lecteur
Flash 2 et versions suivantes.

Voir aussi
« nextScene » à la page 329

print

Syntaxe
```
print (target, "bmovie");
print (target, "bmax");
print (target, "bframe");
```

Arguments
target Le nom d'occurrence du clip à imprimer. Par défaut, toutes les images de l'animation sont imprimées. Si vous ne souhaitez imprimer que des images spécifiques de l'animation, désignez les images à imprimer en associant une étiquette d'image #P à ces images dans l'environnement de programmation.

bmovie Désigne le cadre de délimitation d'une image spécifique dans une animation comme zone d'impression pour toutes les images imprimables de l'animation. Attachez une étiquette #b (dans l'environnement de programmation) pour désigner l'image dont le cadre de délimitation servira de zone d'impression.

bmax Désigne un composite de tous les cadres de délimitation, de toutes les images imprimables, comme zone d'impression. Spécifiez l'argument *bmax* lorsque les images imprimables de votre animation varient en taille.

bframe Indique que le cadre de délimitation de chaque image imprimable est utilisé comme zone d'impression pour cette image. Cela modifie la zone d'impression pour chaque image et modifie la taille des objets pour qu'ils correspondent à la zone d'impression. Utilisez *bframe* si vous avez des objets de tailles diverses dans chaque image et si vous souhaitez que chaque objet remplisse la page imprimée.

Description

Action ; imprime le clip *target* en fonction du modificateur d'imprimante spécifié dans l'argument. Si vous ne souhaitez imprimer que des images spécifiques de l'animation cible, associez une étiquette d'image `#P` aux images que vous souhaitez imprimer. Bien que l'action `print` fournissent des impressions de meilleure qualité que l'action `printAsBitmap`, elle ne peut pas être utilisée pour imprimer des animations utilisant les transparences alpha ou des effets de couleur spéciaux.

Si vous ne spécifiez pas d'argument de zone d'impression, la zone d'impression est déterminée par la taille de la scène de l'animation chargée, par défaut. L'animation n'hérite pas de la taille de scène de l'animation principale. Vous pouvez contrôler la zone d'impression en spécifiant les arguments *bmovie*, *bmax* ou *bframe*.

Tous les éléments imprimables d'une animation doivent être complètement chargés avant de commencer l'impression.

La fonction d'impression de Flash Player supporte les imprimantes PostScript et non PostScript. Les imprimantes non PostScript convertissent les vecteurs en bitmaps.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant imprimera toutes les images imprimables de `myMovie` avec la zone d'impression définie par le cadre de délimitation de l'image portant l'étiquette d'image `#b`.

```
print("myMovie","bmovie");
```

L'exemple suivant imprimera toutes les images imprimables de `myMovie` avec une zone d'impression définie par le cadre de délimitation de chaque image.

```
print("myMovie","bframe");
```

Voir aussi

« printAsBitmap » à la page 348

printAsBitmap

Syntaxe

```
printAsBitmap(target, "bmovie");
printAsBitmap(target, "bmax");
printAsBitmap(target, "bframe");
```

Arguments

target Le nom d'occurrence du clip à imprimer. Par défaut, toutes les images de l'animation sont imprimées. Si vous ne souhaitez imprimer que des images spécifiques d'une animation, désignez les images à imprimer en associant une étiquette d'image #P à ces images dans l'environnement de programmation.

bmovie Désigne le cadre de délimitation d'une image spécifique dans une animation comme la zone d'impression de toutes les images imprimables de l'animation. Attachez une étiquette #b (dans l'environnement de programmation) pour désigner l'image dont vous souhaitez utiliser le cadre de délimitation comme zone d'impression.

bmax Désigne un composite de tous les cadres de délimitation, de toutes les images imprimables, comme la zone d'impression. Spécifiez l'argument *bmax* lorsque les images de votre animation varient en taille.

bframe Indique que le cadre de délimitation de chaque image imprimable est utilisé comme zone d'impression pour cette image. Cela modifie la zone d'impression pour chaque image et modifie la taille des objets pour qu'ils correspondent à la zone d'impression. Utilisez *bframe* si vous avez des objets de tailles diverses dans chaque image et si vous souhaitez que chaque objet remplisse la page imprimée.

Description

Action ; imprime le clip *target* en bitmap. Utilisez `printAsBitmap` pour imprimer des animations contenant des objets qui utilisent des effets de transparence ou de couleur. L'action `printAsBitmap` imprime avec la résolution la plus élevée disponible sur l'imprimante pour préserver autant de définition et de qualité que possible . Pour calculer la taille de fichier imprimable d'une image à imprimer en bitmap, multipliez la largeur en pixels, par la hauteur en pixels, par la résolution de l'imprimante.

Si votre animation ne contient pas de transparences alpha, ni d'effets de couleur, il est recommandé d'utiliser l'action `print` pour un résultat de meilleure qualité.

Par défaut, la zone d'impression est déterminée par la taille de la scène de l'animation chargée. L'animation n'hérite pas de la taille de scène de l'animation principale. Vous pouvez contrôler la zone d'impression en spécifiant les arguments *bmovie*, *bmax* ou *bframe*.

Tous les éléments imprimables d'une animation doivent être complètement chargés avant de commencer l'impression.

La fonction d'impression de Flash Player supporte les imprimantes PostScript et non PostScript. Les imprimantes non PostScript convertissent les vecteurs en bitmaps.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« print » à la page 346

_quality

Syntaxe
```
_quality
_quality = x;
```

Arguments
x Une chaîne spécifiant une des valeurs suivantes :

LOW Qualité de rendu faible. Les graphiques ne sont pas antialiasés et les bitmaps ne sont pas lissés.

MEDIUM Qualité de rendu moyenne. Les graphiques sont antialiasés avec une grille 2x2 mais les bitmaps ne sont pas lissés. Convient pour des animations ne contenant pas de texte.

HIGH Qualité de rendu élevée. Les graphiques sont antialiasés avec une grille 4x4 et les bitmaps sont lissés si l'animation est statique. Il s'agit du paramètre de qualité de rendu par défaut utilisé par Flash.

BEST Qualité de rendu très élevée. Les graphiques sont antialiasés avec une grille 4x4 et les bitmaps sont toujours lissés.

Description
Propriété (globale) ; définit ou extrait la qualité de rendu utilisée pour une animation.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant définit le rendu de `oldQuality` sur HIGH.

```
oldQualtiy = _quality
_quality = "HIGH";
```

Voir aussi
« _highquality » à la page 278

random

Syntaxe
```
random();
```

Arguments
value L'entier le plus grand pour lequel `random` renverra une valeur.

Description
Fonction ; renvoie un entier aléatoire entre 0 et l'entier spécifié dans l'argument *value*.

Lecteur
Flash 4. Cette fonction est désapprouvée dans Flash 5 ; l'utilisation de la méthode `Math.random` est recommandée.

Exemple
L'utilisation suivante de `random` renvoie une valeur de 0, 1, 2, 3, ou 4.

```
random(5);
```

Voir aussi
« Math.random » à la page 306

removeMovieClip

Syntaxe
```
removeMovieClip(target);
```

Arguments
target Le chemin cible d'une occurrence de clip créée avec `duplicateMovieClip`, ou le nom d'occurrence d'un clip créé avec les méthodes `attachMovie` ou `duplicateMovie` de l'objet ClipAnimation.

Description
Action ; supprime une occurrence de clip créée avec les méthodes `attachMovie` ou `duplicateMovieClip` de l'objet ClipAnimation, ou avec l'action `duplicateMovieClip`.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« duplicateMovieClip » à la page 263
« MovieClip.duplicateMovieClip » à la page 314
« MovieClip.attachMovie » à la page 314
« MovieClip.removeMovieClip » à la page 324

return

Syntaxe

```
return[expression];

return;
```

Arguments

expression Un type, une chaîne, un nombre, un tableau ou un objet à évaluer et à renvoyer comme valeur de la fonction. Cet argument est optionnel.

Description

Action ; spécifie la valeur renvoyée par une fonction. Lorsque l'action return est exécutée, l'*expression* est évaluée et renvoyée comme valeur de la fonction. L'action return oblige la fonction à arrêter l'exécution. Si l'instruction `return` est utilisée seule, ou si Flash ne rencontre pas d'instruction `return` pendant une action de boucle, elle renvoie `null`.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `return` :

```
function sum(a, b, c){
return a + b + c;
}
```

Voir aussi

« function » à la page 271

_root

Syntaxe

```
_root;

_root.movieClip;

_root.action;
```

Arguments

movieClip Le nom d'occurrence d'un clip.

action La valeur définie pour la propriété. Cet argument est optionnel et il n'est pas nécessaire de le définir en fonction de la propriété.

Description
Propriété ; spécifie ou renvoie une référence au scénario de l'animation racine. Si une animation possède plusieurs niveaux, le scénario de l'animation racine se situe dans le niveau contenant l'exécution du script courant. Par exemple, si un script de niveau 1 évalue _root, niveau 1 est renvoyé.

Spécifier `_root` est identique à l'utilisation de la notation à barre oblique "/ " pour spécifier un chemin absolu au sein du niveau courant.

Lecteur
Flash 4 et versions suivantes.

Exemple
L'exemple suivant arrête le scénario du niveau contenant l'exécution du script courant.

```
_rootl.stop();
```

L'exemple suivant envoie le scénario dans le niveau courant à l'image 3.

```
_root.gotoAndStop(3);
```

Voir aussi
« _parent » à la page 342
« targetPath » à la page 379

_rotation

Syntaxe
```
instancename._rotation
instancename._rotation = integer
```

Arguments
integer Le nombre de degrés de rotation du clip.

instancename Le clip à pivoter.

Description
Propriété ; spécifie la rotation du clip en degrés.

Lecteur
Flash 4 et versions suivantes.

scroll

Syntaxe

variable_name.scroll = *x*

Arguments

variable_name Le nom d'une variable associée à un champ de texte.

x Le numéro de ligne de la ligne supérieure visible dans le champ de texte. Vous pouvez spécifier cette valeur ou utiliser la valeur par défaut 1. Le Flash Player met à jour cette valeur au fur et à mesure que l'utilisateur fait défiler le champ de texte vers le haut ou le bas.

Description

Propriété ; contrôle l'affichage des informations d'un champ de texte associé à une variable. La propriété `scroll` définit l'endroit où le champ de texte commence à afficher le contenu ; après l'avoir définie, le Flash Player la met à jour au fur et à mesure que l'utilisateur fait défiler le champ de texte. La propriété `scroll` est utile pour diriger les utilisateurs vers un paragraphe spécifique dans un long passage, ou pour créer des champs de texte défilants. Cette propriété peut être extraite et modifiée.

Lecteur

Flash 4 et versions suivantes.

Voir aussi

« maxscroll » à la page 309

Selection (objet)

L'objet Sélection vous permet de définir et de contrôler le champ de texte modifiable ciblé courant. Le champ de texte modifiable ciblé courant est le champ où se trouve actuellement le curseur de l'utilisateur. Les index de la zone de sélection sont basés sur zéro (où la première position est 0, la deuxième est 1, et ainsi de suite).

Il n'existe pas de méthode constructeur pour l'objet Sélection car il ne peut y avoir qu'un seul champ de texte ciblé courant à la fois.

Tableau des méthodes de l'objet Sélection

Méthode	Description
`getBeginIndex`	Renvoie l'index du début de la zone de sélection. Renvoie –1 s'il n'y a pas d'index ou de champ sélectionné courant.
`getCaretIndex`	Renvoie la position courante du point d'insertion dans la zone de sélection ciblée courante. Renvoie -1 s'il n'y a pas de position de point d'insertion ou de zone de sélection ciblée courante.
`getEndIndex`	Renvoie l'index de la fin de la zone de sélection. Renvoie –1 s'il n'y a pas d'index ou de champ sélectionné courant.
`getFocus`	Renvoie le nom de la variable pour le champ de texte modifiable ciblé courant. Renvoie null s'il n'y a pas de champ de texte modifiable ciblé courant.
`setFocus`	Cible le champ de texte modifiable associé à la variable spécifiée dans l'argument.
`setSelection`	Définit les index de début et de fin de la zone de sélection.

Selection.getBeginIndex

Syntaxe

```
Selection.getBeginIndex();
```

Arguments

Aucun.

Description

Méthode ; renvoie l'index du début de la zone de sélection. Si aucun index n'existe ou si aucun champ courant n'est ciblé, la méthode renvoie -1. Les index de la zone de sélection sont basés sur zéro (où la première position est 0, la deuxième est 1, et ainsi de suite).

Lecteur

Flash 5 et versions suivantes.

Selection.getCaretIndex

Syntaxe

```
Selection.getCaretIndex();
```

Arguments

Aucun.

Description
Méthode ; renvoie l'index de la position du curseur clignotant. Si aucun curseur clignotant n'est affiché, la méthode renvoie -1. Les index de la zone de sélection sont basés sur zéro (où la première position est 0, la deuxième est 1, et ainsi de suite).

Lecteur
Flash 5 et versions suivantes.

Selection.getEndIndex

Syntaxe
```
Selection.getEndIndex();
```

Arguments
Aucun.

Description
Méthode ; renvoie l'index de fin de la zone de sélection ciblée courante. S'il n'existe aucun index, ou s'il n'y a pas de zone de sélection ciblée courante, la méthode renvoie -1. Les index de la zone de sélection sont basés sur zéro (où la première position est 0, la deuxième est 1, et ainsi de suite).

Lecteur
Flash 5 et versions suivantes.

Selection.getFocus

Syntaxe
```
Selection.getFocus();
```

Arguments
Aucun.

Description
Méthode ; renvoie le nom de la variable du champ de texte modifiable ciblé courant. Si aucun champ de texte n'est actuellement ciblé, la méthode renvoie `null`.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant renvoie le nom de la variable.
```
_root.anyMovieClip.myTextField.
```

Selection.setFocus

Syntaxe

```
Selection.setFocus(variable);
```

Arguments

variable Une chaîne spécifiant le nom d'une variable associée à un champ de texte avec la notation à point ou à barre oblique.

Description

Méthode ; cible le champ de texte modifiable associée à la *variable* spécifiée.

Lecteur

Flash 5 et versions suivantes.

Selection.setSelection

Syntaxe

```
Selection.setSelection(start, end);
```

Arguments

start L'index de début de la zone de sélection.

end L'index de fin de la zone de sélection.

Description

Méthode ; définit la zone de sélection du champ de texte ciblé courant. La nouvelle zone de sélection commencera à l'index spécifié dans l'argument *start* et finira à l'index spécifié dans l'argument *end*. Les index de la zone de sélection sont basés sur zéro (où la première position est 0, la deuxième est 1, et ainsi de suite). Cette méthode n'a aucun effet s'il n'y a pas de champ de texte ciblé courant.

Lecteur

Flash 5 et versions suivantes.

set

Syntaxe

```
variable = expression;
set(variable, expression);
```

Arguments

variable Le nom du conteneur où se trouve la valeur de l'argument *expression*.

expression La valeur (ou une séquence qui peut être évaluée à une valeur) affectée à la variable.

Description

Action ; affecte une valeur à une variable. Une variable est un conteneur qui stocke des informations. Le conteneur reste toujours le même, c'est le contenu qui peut varier. En modifiant la valeur d'une variable pendant la lecture d'une animation, vous pouvez enregistrer et sauvegarder les informations relatives aux actions de l'utilisateur, enregistrer les valeurs modifiées pendant la lecture de l'animation ou évaluer si une condition est `true` ou `false`.

Les variables peuvent stocker des nombres ou des chaînes de caractères. Chaque animation et chaque clip possèdent leur propre jeu de variables et chaque variable possède sa propre valeur indépendante des variables des autres animations ou clips.

ActionScript est un langage non typé. Cela signifie que les variables n'ont pas besoin d'être explicitement définies comme contenant un nombre ou une chaîne. Flash interprète le type de données selon qu'il s'agit d'un entier ou d'une chaîne.

Utilisez l'instruction `set` en conjonction avec l'action `call` pour transmettre ou renvoyer des valeurs.

Lecteur

Flash 4 et versions suivantes.

Exemple

Cet exemple définit une variable appelée `orig_x_pos` qui stocke la position originale de l'axe *x* du clip `ship` afin de pouvoir redéfinir ship sur sa position de départ plus tard dans l'animation.

```
on(release) {
    set(x_pos, getProperty ("ship", _x ));
}
```

Cet exemple aurait pu être écrit ainsi :

```
on(release) {
   orig_x_pos = getProperty ("ship", _x );
}
```

Voir aussi

« var » à la page 387
« call » à la page 236

setProperty

Syntaxe

```
setProperty(target, property, expression);
```

Arguments

target Le chemin du nom d'occurrence du clip dont la propriété est définie.

property La propriété à définir.

expression La valeur sur laquelle la propriété est définie.

Description

Action ; modifie la propriété d'un clip pendant la lecture de l'animation.

Lecteur

Flash 4 et versions suivantes.

Exemple

Cette instruction définit la propriété `_alpha` du clip nommé `star` sur 30 pour cent lorsque le bouton est cliqué.

```
on(release) {
    setProperty("star", _alpha, 30);
}
```

Voir aussi

« getProperty » à la page 273

Son (objet)

L'objet Son vous permet de définir et de contrôler les sons dans une occurrence de clip particulière ou pour le scénario global, si vous ne spécifiez pas d'argument *target* lors de la création du nouvel objet Son. Vous devez utiliser le constructeur `new Sound` pour créer une occurrence de l'objet Son avant d'appeler les méthodes de l'objet Son.

L'objet Son n'est pris en charge que par la version 5 de Flash Player.

Tableau des méthodes de l'objet Son

Méthode	Description
`attachSound`	Attache le son spécifié dans l'argument.
`getPan`	Renvoie la valeur de l'appel précédent setPan.
`getTransform`	Renvoie la valeur de l'appel précédent setTransform.
`getVolume`	Renvoie la valeur de l'appel précédent setVolume.
`setPan`	Définit la balance gauche/droite du son.
`setTransform`	Définit les transformations pour un son.
`setVolume`	Définit le niveau du volume pour un son.
`start`	Démarre la lecture d'un son depuis le début ou, en option, depuis un point de décalage défini dans l'argument.
`stop`	Arrête le son spécifié ou tous les sons en cours de lecture.

Constructeur de l'objet Son

Syntaxe

```
new Sound();
new Sound(target);
```

Arguments

target L'occurrence de clip à laquelle l'objet Son est appliqué. Cet argument est optionnel.

Description

Méthode ; crée un nouvel objet Son pour un clip spécifié. Si vous ne spécifiez pas de *target*, l'objet Son contrôlera tous les sons dans le scénario global.

Lecteur

Flash 5 et versions suivantes.

Exemple

```
GlobalSound = new Sound();
MovieSound = new Sound(mymovie);
```

Sound.attachSound

Syntaxe

```
mySound.attachSound("idName");
```

Arguments

idName Le nom pour la nouvelle occurrence du son. C'est le même nom que celui entré comme identifiant dans la boîte de dialogue Propriétés de liaison de symbole. Cet argument doit être écrit entre " " (guillemets).

Description

Méthode ; associe le son spécifié dans l'argument *idName* à l'objet Son spécifié. Le son doit se trouver dans la bibliothèque de l'animation courante et être spécifié pour l'export dans la boîte de dialogue Propriétés de liaison de symbole. Vous devez appeler `Sound.start` pour démarrer la lecture du son.

Lecteur

Version 5 ou versions ultérieures de Flash.

Voir aussi

« Sound.start » à la page 366

Sound.getPan

Syntaxe

```
mySound.getPan();
```

Arguments

Aucun.

Description

Méthode ; renvoie le niveau d'étendue défini dans le dernier appel `setPan` sous forme d'entier entre -100 et 100. Le paramètre d'étendue contrôle la balance gauche-droite des sons courants et futurs d'une animation.

Cette méthode est cumulable avec les méthodes `setVolume` ou `setTransform`.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« Sound.setPan » à la page 361

« Sound.setTransform » à la page 363

Sound.getTransform

Syntaxe
mySound.getTransform();

Arguments
Aucun.

Description
Méthode ; renvoie les informations de transformation du son pour l'objet Son spécifié défini avec le dernier appel `setTransform`.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« Sound.setTransform » à la page 363

Sound.getVolume

Syntaxe
mySound.getVolume();

Arguments
Aucun.

Description
Méthode ; renvoie le niveau du volume du son sous la forme d'un entier entre 0 et 100, avec 0 pour éteint et 100 pour volume maximum. Le paramètre par défaut est 100.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« Sound.setVolume » à la page 365

Sound.setPan

Syntaxe
mySound.setPan(*pan*);

Arguments
pan Un entier spécifiant la balance gauche-droite d'un son. Les valeurs correctes sont comprises entre -100 et 100, avec -100 pour le canal de gauche, 100 pour le canal de droite et 0 pour répartir le son d'une manière uniforme entre les deux canaux.

Description

Méthode ; détermine comment le son est réparti dans les canaux droit et gauche (haut-parleurs). Pour les sons mono, *pan* affecte le haut-parleur (gauche ou droit) par lequel passe le son.

Cette méthode est cumulable avec les méthodes `setVolume` et `setTransform` et appeler cette méthode supprime et met à jour les paramètres `setPan` et `setTransform` précédents.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `setVolume` et `setPan` pour contrôler un objet Son avec la cible spécifiée « u2 ».

```
onClipEvent(mouseDown) {

// create a sound object and

s = new Sound(this);

// attach a sound in the library

s.attachSound("u2");

//set volume at 50%

s.setVolume(50);

//turn off the sound in the right channel

s.setPan(-100);

//start 30 seconds into the sound and play it 5 times

s.start(30, 5);
```

Voir aussi

« Sound.setTransform » à la page 363

« Sound.setVolume » à la page 365

Sound.setTransform

Syntaxe

mySound.setTransform(*soundTransformObject*);

Arguments

soundTransformObject Un objet créé avec le constructeur pour l'objet générique Object.

Description

Méthode ; définit les informations de transformation du son pour un objet Son. Cette méthode est cumulable avec les méthodes `setVolume` et `setPan` et appeler cette méthode supprime et met à jour les paramètres `setPan` ou `setVolume` précédents. Cet appel s'adresse aux utilisateurs experts qui souhaitent ajouter des effets sonores intéressants.

Les sons utilisent un espace disque et une mémoire considérables. Du fait que les sons stéréo utilisent deux fois plus de données que les sons mono, il est généralement conseillé d'utiliser des sons mono de 6 bits et 22 kHz. Vous pouvez utiliser la méthode `setTransform` pour lire des sons mono en stéréo, des sons stéréo en mono et pour ajouter des effets sonores intéressants.

L'argument `sound transformObject` est un objet que vous créez avec la méthode constructeur de l'objet générique Objet avec des paramètres spécifiant comment le son est réparti dans les canaux gauche et droit (haut-parleurs).

Les paramètres de l'objet sound transformObject sont les suivants :

`11` Une valeur de pourcentage spécifiant la quantité de l'entrée gauche à lire dans le haut-parleur gauche (-100 à 100).

`1r` Une valeur de pourcentage spécifiant la quantité de l'entrée droite à lire dans le haut-parleur gauche (-100 à 100).

`rr` Une valeur de pourcentage spécifiant la quantité de l'entrée droite à lire dans le haut-parleur droit (-100 à 100).

`rl` Une valeur de pourcentage spécifiant la quantité de l'entrée gauche à lire dans le haut-parleur droit (-100 à 100).

Le résultat net de ces paramètres est représenté par les formules suivantes :

leftOutput = left input * ll + right input * lr

rightOutput = right lnput * rr + left input * rl

Les valeurs de l'entrée gauche ou droite sont déterminées par le type de son (mono ou stéréo) de votre animation.

Les sons stéréo divisent l'entrée de son de manière égale entre les haut-parleurs droit et gauche et ont les paramètres de transformation par défaut suivants :

```
ll = 100
lr = 0
rr = 100
rl = 0
```

Les sons mono lisent toute l'entrée du son dans le haut-parleur gauche et ont les paramètres de transformation par défaut suivants :

```
ll = 100
lr = 100
rr = 0
rl = 0
```

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant crée un objet SoundTransformObject qui lit les deux canaux droit et gauche dans le canal gauche.

```
mySoundTransformObject = new Object
mySoundTransformObject.ll = 100
mySoundTransformObject.lr = 100
mySoundTransformObject.rr = 0
mySoundTransformObject.rl = 0
```

Le code précédent crée un objet SoundTransformObject. Pour l'appliquer à un objet Son, vous devez transmettre l'objet à un objet Son en utilisant `setTransform` ainsi :

```
mySound.setTransform(mySoundTransformObject);
```

Les exemples de paramètres suivants peuvent être définis en utilisant `setTransform`, mais ne peuvent pas être définis avec `setVolume` ou `setPan`, même en combinaison.

Ce code lit les deux canaux gauche et droit par le canal gauche :

```
mySound.setTransform(soundTransformObjectLeft);
```

Dans le code précédent, *`soundTransformObjectLeft`* possède les paramètres suivants :

```
11 = 100
1r = 100
rr = 0
rl = 0
```

Exemple
Ce code lit un son stéréo en mono :

```
setTransform(soundTransformObjectMono);
```

Dans le code précédent, *soundTransformObjectMono* possède les paramètres suivants :

```
ll = 50
lr = 50
rr = 50
rl = 50
```

Exemple
Ce code lit le canal gauche à la moitié de sa capacité et ajoute le reste du canal gauche dans le canal droit :

```
setTransform(soundTransformObjectHalf);
```

Dans le code précédent, *soundTransformObjectHalf* possède les paramètres suivants :

```
ll = 50
lr = 0
rr = 100
rl = 50
```

Voir aussi
« Constructeur pour l'objet Objet » à la page 336

Sound.setVolume

Syntaxe
mySound.setVolume(*volume*);

Arguments
volume Un nombre entre 0 et 100 représentant le niveau du volume. 100 est le volume maximum et 0 représente le volume éteint. Le paramètre par défaut est 100.

Description
Méthode ; définit le volume pour l'objet Son.

Cette méthode est cumulable avec les méthodes `setPan` et `setTransform`.

Lecteur
Flash 5 et versions suivantes.

Exemple

L'exemple suivant définit le volume sur 50% et transfert le son dans le temps du haut-parleur gauche vers le droit.

```
onClipEvent (load) {
   i = -100;
   s = new sound();
   s.setVolume(50);
}
onClipEvent (enterFrame) {
   S.setPan(i++);
}
```

Voir aussi

« Sound.setPan » à la page 361

« Sound.setTransform » à la page 363

Sound.start

Syntaxe

```
mySound.start();

mySound.start([secondOffset, loop]);
```

secondOffset Un argument optionnel vous permettant de démarrer la lecture du son en un point spécifique. Par exemple, si vous avez un son de 30 secondes et que vous souhaitez démarrer la lecture du son au milieu, spécifiez 15 dans l'argument *secondOffset*. Le son n'est pas retardé de 15 secondes, mais la lecture démarrera à la seconde 15.

loop Un argument optionnel vous permettant de spécifier le nombre de répétitions du son.

Description

Méthode ; démarre la lecture du dernier son associé, depuis le début si aucun argument n'est spécifié, ou à l'endroit du son spécifié dans l'argument *secondOffset*.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« Sound.setPan » à la page 361

« Sound.stop » à la page 367

Sound.stop

Syntaxe

```
mySound.stop();
mySound.stop(["idName"]);
```

Arguments

idName Un argument optionnel spécifiant le son spécifique à arrêter. L'argument *idName* doit se trouver entre guillemets (" ").

Description

Méthode ; arrête tous les sons en cours de lecture si aucun argument n'est spécifié, ou uniquement le son spécifié dans l'argument *idName*.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« Sound.start » à la page 366

_soundbuftime

Syntaxe

```
_soundbuftime = integer;
```

Arguments

integer Le nombre de secondes avant que l'animation ne démarre en lecture continue.

Description

Propriété (globale) ; établit le nombre de secondes de son en lecture continue dans le pré-tampon. La valeur par défaut est de 5 secondes.

Lecteur

Flash 4 et versions suivantes.

startDrag

Syntaxe

```
startDrag(target);
startDrag(target,[lock]);
startDrag(target [,lock [,left , top , right, bottom]]);
```

Arguments

target Le chemin cible du clip à faire glisser.

lock Une valeur booléenne spécifiant si le clip glissable est verrouillé au centre de la position de la souris (*true*) ou verrouillé sur le point où l'utilisateur a cliqué la première fois dans le clip (*false*). Cet argument est optionnel.

left, top, right, bottom Valeurs relatives aux coordonnées du parent du clip spécifiant un rectangle contraint pour le clip. Ces arguments sont optionnels.

Description

Action ; rend le clip *target* glissable pendant la lecture de l'animation. Un seul clip peut être glissé à la fois. Une fois l'opération `startDrag` exécutée, le clip reste glissable jusqu'à ce qu'il soit explicitement arrêté par une action `stopDrag` ou jusqu'à ce qu'une action `startDrag` soit appelée pour un autre clip.

Exemple

Pour créer un clip que l'utilisateur pourra positionner à n'importe quel endroit, associez les actions `startDrag` et `stopDrag` à un bouton dans le clip, comme dans l'exemple suivant :

```
on(press) {
    startDrag("",true);
}
on(release) {
    stopDrag();
}
```

Voir aussi

« stopDrag » à la page 369

« _droptarget » à la page 262

stop

Syntaxe

```
stop;
```

Arguments

Aucun.

Description
Action ; arrête l'animation en cours de lecture. L'utilisation la plus fréquente de cette action sert à contrôler les clips à l'aide de boutons.

Lecteur
Flash 3 et versions suivantes.

stopAllSounds

Syntaxe
```
stopAllSounds();
```

Arguments
Aucun.

Description
Action ; arrête tous les sons en cours de lecture dans une animation sans arrêter la tête de lecture. Les sons définis en continu reprendront la lecture lorsque la tête de lecture passera dans les images où ils se trouvent.

Lecteur
Flash 3 et versions suivantes.

Exemple
Le code suivant peut être appliqué à un bouton qui, lorsqu'il est cliqué, arrête tous les sons de l'animation.

```
on(release) {
    stopAllSounds();
}
```

Voir aussi
« Son (objet) » à la page 358

stopDrag

Syntaxe
```
stopDrag();
```

Arguments
Aucun.

Description
Action ; arrête l'opération Drag courante.

Lecteur
Flash 4 et versions suivantes.

Exemple
Cette instruction arrête l'action Drag de l'occurrence `mc` lorsque l'utilisateur relâche le bouton de la souris.

```
on(press) {
   startDrag("mc");
}
on(release) {
 stopdrag();
}
```

Voir aussi
« startDrag » à la page 368
« _droptarget » à la page 262

String (fonction)

Syntaxe
`String(`*`expression`*`);`

Arguments
`expression` Le nombre, le booléen, la variable ou l'objet à convertir en chaîne.

Description
Fonction ; renvoie une représentation chaîne de l'argument spécifié selon les règles suivantes :

Si *x* est booléen, la chaîne renvoyée est `true` ou `false`.

Si *x* est un nombre, la chaîne renvoyée est une représentation décimale du nombre.

Si *x* est une chaîne, la chaîne renvoyée est `x`.

Si *x* est un objet, la valeur renvoyée est la représentation d'une chaîne de l'objet, générée par l'appel de la propriété String de l'objet ou par l'appel de `object.toString`, si une telle propriété n'existe pas.

Si *x* est un clip, la valeur renvoyée est le chemin cible du clip en notation à barre oblique (*/*).

Si *x* est indéfini, la valeur renvoyée est une chaîne vide.

Lecteur
Flash 3 et versions suivantes.

Voir aussi
« Object.toString » à la page 337
« Number.toString » à la page 335
« Chaîne (objet) » à la page 371
« " " (délimiteur de chaîne) » à la page 371

" " (délimiteur de chaîne)

Syntaxe
"*text*"

Arguments
text Tout texte.

Description
Délimiteur de chaîne ; lorsqu'ils sont utilisés avant et après une chaîne, les guillemets indiquent que la chaîne est un caractère (pas une variable, ni une valeur numérique, ni un autre élément ActionScript).

Lecteur
Flash 4 et versions suivantes.

Exemple
Cette instruction utilise les guillemets pour indiquer que la chaîne “Prince Edward Island” est une chaîne de caractères et non la valeur d'une variable :

```
province = "Prince Edward Island"
```

Voir aussi
« Chaîne (objet) » à la page 371
« String (fonction) » à la page 370

Chaîne (objet)

L'objet Chaîne est une enveloppe pour le type de données primitif string, ce qui vous permet d'utiliser les méthodes et les propriétés de l'objet Chaîne pour manipuler les types de valeurs de chaîne primitifs. Vous pouvez convertir la valeur de n'importe quel objet en chaîne avec la fonction `String()`.

Toutes les méthodes de l'objet Chaîne, à l'exception de `concat`, `fromCharCode`, `slice` et `substr`, sont génériques. Cela signifie que les méthodes appellent elles-mêmes `this.toString` avant d'effectuer leurs opérations et que vous pouvez utiliser ces méthodes avec des objets autres que Chaîne.

Vous pouvez appeler toutes les méthodes de l'objet Chaîne en utilisant la méthode constructeur `new String` ou en utilisant une valeur de chaîne littérale. Si vous spécifiez une chaîne littérale, l'interprète ActionScript la convertit automatiquement en objet Chaîne temporaire, appelle la méthode puis supprime l'objet Chaîne temporaire. Vous pouvez également utiliser la propriété `String.length` avec une chaîne littérale.

Il est important de ne pas faire de confusion entre une chaîne littérale et une occurrence de l'objet Chaîne. Dans l'exemple suivant, la première ligne du code crée la chaîne littérale `s1` et la seconde ligne du code crée une occurrence de l'objet Chaîne `s2`.

```
s1 = "foo"
s2 = new String("foo")
```

Il est recommandé d'utiliser des chaînes littérales, à moins d'avoir spécifiquement besoin d'un objet Chaîne, car les objets Chaîne peuvent avoir un comportement allant à l'encontre de l'intuition.

Tableau des méthodes de l'objet Chaîne

Méthode	Description
`charAt`	Renvoie un nombre correspondant à la position du caractère dans la chaîne.
`charCodeAt`	Renvoie la valeur du caractère situé à l'index donné sous forme d'entier de 16 bits entre 0 et 65535.
`concat`	Combine le texte de deux chaînes et renvoie une nouvelle chaîne.
`fromCharCode`	Renvoie une chaîne constituée des caractères spécifiés dans les arguments.
`indexOf`	Recherche la chaîne et renvoie l'index de la valeur spécifiée dans les arguments. Si la valeur apparaît plus d'une fois, l'index de la première occurrence est renvoyé. Si la valeur n'est pas trouvée, -1 est renvoyé.
`lastIndexOf`	Renvoie la dernière occurrence de sous-chaîne dans la chaîne, qui apparaît avant la position de départ spécifiée dans l'argument, ou -1 s'il n'y en a pas.
`slice`	Extrait une section d'une chaîne et renvoie une nouvelle chaîne.
`split`	Fractionne un objet Chaîne en tableau de chaînes en découpant la chaîne en sous-chaînes.
`substr`	Renvoie un nombre spécifié de caractères d'une chaîne, en commençant à l'endroit spécifié dans l'argument.
`substring`	Renvoie les caractères entre deux index, spécifiés dans les arguments, dans la chaîne.
`toLowerCase`	Convertit la chaîne en lettres minuscules et renvoie le résultat.
`toUpperCase`	Convertit la chaîne en lettres majuscules et renvoie le résultat.

Tableau des propriétés de l'objet Chaîne

Propriété	Description
`length`	Renvoie la longueur de la chaîne.

Constructeur de l'objet Chaîne

Syntaxe
```
new String(value);
```

Arguments
value La valeur initiale du nouvel objet Chaîne.

Description
Constructeur ; crée un nouvel objet Chaîne.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« String (fonction) » à la page 370
« " " (délimiteur de chaîne) » à la page 371

String.charAt

Syntaxe
```
myString.charAt(index);
```

Arguments
index Le numéro du caractère de la chaîne à renvoyer.

Description
Méthode ; renvoie le caractère spécifié par l'argument *index*. L'index du premier caractère d'une chaîne est 0. Si *index* n'est pas un nombre compris entre 0 et `string.length` - 1, une chaîne vide est renvoyée.

Lecteur
Flash 5 et versions suivantes.

String.charCodeAt

Syntaxe
myString.charCodeAt(*index*);

Arguments
index Le numéro du caractère pour lequel la valeur est extraite.

Description
Méthode ; renvoie la valeur du caractère spécifié par *index*. La valeur renvoyée est un entier de 16 bits compris entre 0 et 65535.

Cette méthode est semblable à string.charAt excepté que la valeur renvoyée concerne le caractère d'un emplacement précis, au lieu d'une sous-chaîne contenant le caractère.

Lecteur
Flash 5 et versions suivantes.

String.concat

Syntaxe
myString.concat(*value1,...valueN*);

Arguments
value1,...valueN Zéro ou plusieurs valeurs à concaténer.

Description
Méthode ; combine les valeurs spécifiées et renvoie une nouvelle chaîne. Si nécessaire, chaque argument *value* est converti en chaîne et ajouté à la fin de la chaîne, dans l'ordre.

Lecteur
Flash 5 et versions suivantes.

String.fromCharCode

Syntaxe
myString.fromCharCode(*c1,c2,...cN*);

Arguments
c1,c2,...cN Les caractères à transformer en chaîne.

Description
Méthode ; renvoie une chaîne constituée des caractères spécifiés dans les arguments.

Lecteur
Flash 5 et versions suivantes.

String.indexOf

Syntaxe

```
myString.indexOf(value);

myString.index of (value, start);
```

Arguments

value Un entier ou une chaîne spécifiant la sous-chaîne à rechercher dans *myString*.

start Un entier spécifiant le point de départ de la sous-chaîne. Cet argument est optionnel.

Description

Méthode ; recherche la chaîne et renvoie la position de la première occurrence de l'argument *value* spécifié. Si la valeur n'est pas trouvée, la méthode renvoie -1.

Lecteur

Flash 5 et versions suivantes.

String.lastIndexOf

Syntaxe

```
myString.lastIndexOf(substring);

myString.lastIndexOf(substring, start);
```

Arguments

substring Un entier ou une chaîne spécifiant la chaîne à rechercher.

start Un entier spécifiant le point de départ à l'intérieur de la sous-chaîne. Cet argument est optionnel.

Description

Méthode ; recherche la chaîne et renvoie l'index de la dernière occurrence de `substring` trouvée dans la chaîne appelée. Si *substring* n'est pas trouvée, la méthode renvoie -1.

Lecteur

Flash 5 et versions suivantes.

String.length

Syntaxe

```
string.length
```

Arguments

Aucun.

Description
Propriété ; renvoie le nombre de caractères de l'objet Chaîne spécifié. L'index du dernier caractère d'une chaîne `x` est x.length-1.

Lecteur
Flash 5 et versions suivantes.

String.slice

Syntaxe
```
myString.slice(start, end);
```

Arguments
start Un nombre spécifiant l'index du point de départ de la section. Si *start* est un nombre négatif, le point de départ est déterminé à partir de la fin de la chaîne, -1 étant le dernier caractère.

end Un nombre spécifiant l'index du point d'arrivée de la section. Si *end* n'est pas spécifié, la section inclut tous les caractères du début jusqu'à la fin de la chaîne. Si *end* est un nombre négatif, le point d'arrivée est déterminé depuis la fin de la chaîne, -1 étant le dernier caractère.

Description
Méthode ; extrait une section, ou une sous-chaîne, de l'objet Chaîne spécifié ; puis la renvoie sous forme de nouvelle chaîne, sans modifier l'objet Chaîne d'origine. La chaîne renvoyée inclut le caractère *start* et tous les caractères suivants jusqu'au caractère *end* (exclu).

Lecteur
Flash 5 et versions suivantes.

String.split

Syntaxe
```
myString.split(delimiter);
```

Arguments
delimiter Le caractère utilisé pour délimiter la chaîne.

Description
Méthode ; fractionne un objet Chaîne en cassant la chaîne à l'endroit où se trouve l'argument *delimiter* et renvoie les sous-chaînes dans un tableau. Si aucun délimiteur n'est spécifié, le tableau renvoyé contient un seul élément (la chaîne elle-même). Si le délimiteur est une chaîne vide, chaque caractère de l'objet Chaîne devient un élément dans le tableau.

Lecteur
Flash 5 et versions suivantes.

String.substr

Syntaxe

```
myString.substr(start, length);
```

Arguments

start Un entier indiquant la position du premier caractère dans la sous-chaîne en création. Si *star* test un nombre négatif, la position de départ est déterminée depuis la fin de la chaîne, -1 étant le dernier caractère.

length Le nombre de caractères dans la sous-chaîne en création. Si *length* n'est pas spécifié, la sous-chaîne inclut tous les caractères du début jusqu'à la fin de la chaîne.

Description

Méthode ; renvoie les caractères d'une chaîne depuis l'index spécifié par l'argument *start* jusqu'au nombre de caractères spécifié dans l'argument *length*.

Lecteur

Flash 5 et versions suivantes.

String.substring

Syntaxe

```
myString.substring(from, to);
```

Arguments

from Un entier indiquant la position du premier caractère dans la sous-chaîne en création. Les valeurs correctes pour *from* sont comprises entre 0 et `string.length` - 1.

to Un entier étant égal à1+ l'index du dernier caractère de la sous-chaîne en création. Les valeurs correctes pour *to* sont comprises entre 1 et `string.length`. Si l'argument *to* n'est pas spécifié, la fin de la sous-chaîne correspond à la fin de la chaîne. Si *from* est égal à *to*, la méthode renvoie une chaîne vide. Si *from* est supérieur à *to*, les arguments sont automatiquement permutés avant l'exécution de la fonction.

Description

Méthode ; renvoie une chaîne constituée des caractères contenus entre les points spécifiés par les arguments *from* et *to*.

Lecteur

Flash 5 et versions suivantes.

String.toLowerCase

Syntaxe
```
myString.toLowerCase();
```

Arguments
Aucun.

Description
Méthode ; renvoie une copie de l'objet Chaîne, les caractères majuscules étant convertis en minuscules.

Lecteur
Flash 5 et versions suivantes.

String.toUpperCase

Syntaxe
```
myString.toUpperCase();
```

Arguments
Aucun.

Description
Méthode ; renvoie une copie de l'objet Chaîne, les caractères minuscules étant convertis en majuscules.

Lecteur
Flash 5 et versions suivantes.

substring

Syntaxe
```
substring(string, index, count);
```

Arguments
string La chaîne de laquelle extraire la nouvelle chaîne.

index Le numéro du premier caractère à extraire.

count Le nombre de caractères à inclure dans la chaîne extraite, excluant le caractère d'index.

Description
Fonction de chaîne ; extrait la partie d'une chaîne.

Lecteur
Flash 4 et versions suivantes. Cette fonction a été désapprouvée dans Flash 5.

Voir aussi
« String.substring » à la page 377

_target

Syntaxe
instancename._target

Arguments
instancename Le nom d'une occurrence de clip.

Description
Propriété (lecture seule) ; renvoie le chemin cible de l'occurrence de clip spécifiée dans l'argument *instancename*.

Lecteur
Flash 4 et versions suivantes.

targetPath

Syntaxe
targetpath(*movieClipObject*);

Arguments
movieClipObject La référence (par exemple, _root ou _parent) au clip pour lequel le chemin cible est extrait.

Description
Fonction ; renvoie une chaîne contenant le chemin cible de *movieClipObject*. Le chemin cible est renvoyé avec la notation du point. Pour extraire le chemin cible dans la notation à barre oblique, utilisez la propriété _target.

Lecteur
Flash 5 et versions suivantes.

Exemple
Les exemples suivants sont identiques. Le premier exemple utilise la notation du point et le deuxième exemple utilise la notation à barre oblique.

```
targetPath (Board.Block[index*2+1]) {
play();
}
```

est équivalent à :

```
tellTarget ("Board/Block:" + (index*2+1)) {
play();
}
```

Voir aussi
« eval » à la page 265

tellTarget

Syntaxe

```
tellTarget(target) {
statement;
}
```

Arguments

target Une chaîne de chemin cible spécifiant le scénario à contrôler.

statement Les instructions appliquées au scénario ciblé.

Description

Action ; applique les instructions spécifiées dans l'argument *statements* au scénario spécifié dans l'argument *target*. L'action `tellTarget` est utile pour les contrôles de navigation. Affectez `tellTarget` aux boutons qui arrêtent ou démarrent les clips n'importe où dans la scène. Vous pouvez également faire aller les clips dans une image particulière de ce clip. Par exemple, vous pouvez affecter `tellTarget` à un bouton qui arrête ou démarre les clips sur la scène ou qui dirige les clips dans une image particulière.

L'action `tellTarget` est très semblable à l'action `with`, excepté que `with` prend un clip ou un autre objet comme *target* et `tellTarget` nécessite le chemin cible d'un clip et ne peut pas contrôler les objets.

Lecteur

Flash 3 et versions suivantes. Cette action est désapprouvée dans Flash 5 ; l'utilisation de l'action `with` est recommandée.

Exemple

Cette instruction `tellTarget` contrôle l'occurrence de clip `ball` du scénario principal. L'image 1 du clip est vide et possède une action `stop` invisible sur la scène. Lorsque le bouton de l'action suivante est cliqué, `tellTarget` indique à la tête de lecture du clip `ball` d'aller à l'image 2 et de lire l'animation qui commence là.

```
on(release) {
    tellTarget("ball") {
        gotoAndPlay(2);
    }
}
```

Voir aussi

« with » à la page 390

this

Syntaxe
`this`

Arguments
Aucun.

Description
Mot-clé ; fait référence à un objet ou à l'occurrence d'un clip. Le mot-clé `this` possède le même objectif et la même fonction dans ActionScript que dans JavaScript, avec quelques fonctionnalités supplémentaires. Dans ActionScript, lorsqu'un script est exécuté, `this` fait référence à l'occurrence de clip qui contient le script. Lorsqu'il est utilisé avec l'invocation d'une méthode, `this` contient une référence à l'objet qui contient la méthode exécutée.

Lecteur
Flash 5 et versions suivantes.

Exemple
Dans l'exemple suivant, le mot-clé `this` fait référence à l'objet Cercle.

```
function Circle(radius){
  this.radius = radius;
  this.area = math.PI * radius * radius;
}
```

Dans l'instruction suivante affectée à une image, le mot-clé `this` fait référence au clip courant.

```
//sets the alpha property of the current movie clip to 20.

this._alpha = 20;
```

Dans l'instruction suivante au sein d'un gestionnaire `onClipEvent`, le mot-clé `this` fait référence au clip courant.

```
//when the movie clip loads, a startDrag operation is initiated
for the current movie clip.

onClipEvent (load) {
  startDrag (this, true);
}
```

Voir aussi
« new » à la page 328

toggleHighQuality

Syntaxe
```
toggleHighQuality();
```

Arguments
Aucun.

Description
Action ; active ou désactive l'antialias dans le Flash Player. L'antialias lisse les bords des objets et ralentit la lecture de l'animation. L'action `toggleHighQuality` affecte toutes les animations dans le Flash Player.

Lecteur
Flash 2 et versions suivantes.

Exemple
Le code suivant peut être appliqué à un bouton qui, lorsqu'il est cliqué, active ou désactive l'antialias.

```
on(release) {
    toggleHighQuality();
}
```

Voir aussi
« _quality » à la page 349

« _highquality » à la page 278

_totalframes

Syntaxe
instancename._totalframes

Arguments
instancename Le nom du clip à évaluer.

Description
Propriété (lecture seule) ; évalue le clip spécifié dans l'argument *instancename* et renvoie le nombre total d'images de l'animation.

Lecteur
Flash 4 et versions suivantes.

trace

Syntaxe

`trace(expression);`

Arguments

`expression` Une instruction à évaluer. Lorsque vous testez l'animation, les résultats de l'argument *`expression`* sont affichés dans la fenêtre Résultat.

Description

Action ; évalue l'*`expression`* et affiche les résultats dans la fenêtre Résultat en mode test d'animation.

Utilisez `trace` pour enregistrer des remarques de programmation ou pour afficher des messages dans la fenêtre Résultat pendant le test d'une animation. Utilisez le paramètre *`expression`* pour vérifier si une condition existe ou pour afficher les valeurs dans la fenêtre Résultat. L'action `trace` est identique à la fonction JavaScript `alert`.

Lecteur

Flash 4 et versions suivantes.

Exemple

Cet exemple provient d'un jeu dans lequel une occurrence de clip glissable nommée `rabbi` doit être relâchée sur une cible spécifique. Une instruction conditionnelle évalue la propriété `_droptarget` et exécute différentes actions en fonction de l'endroit où `rabbi` est relâchée. L'action `trace` est utilisée à la fin du script pour évaluer l'emplacement du clip `rabbi` et affiche les résultats dans la fenêtre Résultat. Si `rabbi` ne se comporte pas de la manière prévue (par exemple, s'il atteint la mauvaise cible), les valeurs envoyées dans la fenêtre Résultat par l'action `trace` vous aideront à déterminer le problème dans le script.

```
on(press) {
rabbi.startDrag();
}
on(release) {
if(eval(_droptarget) != target) {
     rabbi._x = rabbi_x;
     rabbi._y = rabbi_y;
} else {
rabbi_x = rabbi._x;
rabbi_y = rabbi._y;
target = "_root.pasture";
}
trace("rabbi_y = " + rabbi_y);
trace("rabbi_x = " + rabbi_x);
stopDrag();
}
```

typeof

Syntaxe

```
typeof(expression);
```

Arguments

expression Une chaîne, un clip, un objet ou une fonction.

Description

Opérateur ; un opérateur unaire placé avant un argument unique. Oblige Flash à évaluer *expression* ; le résultat est une chaîne spécifiant si l'expression est une chaîne, un clip, un objet ou une fonction.

Lecteur

Flash 5 et versions suivantes.

unescape

Syntaxe

```
unescape(x);
```

Arguments

x Une chaîne avec des séquences hexadécimales à échapper.

Description

Fonction de haut niveau ; évalue l'argument *x* comme une chaîne, décode la chaîne d'après un format de code URL (convertissant toutes les séquences hexadécimales en caractères ASCII) et renvoie la chaîne.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant illustre le processus de conversion d'échappement en non-échappement.

```
escape("Hello{[World]}");
```

Le résultat d'échappement est le suivant :

```
("Hello%7B%5BWorld%5D%7D');
```

Utilisez `unescape` pour retourner au format original :

```
unescape("Hello%7B%5BWorld%5D%7D")
```

Le résultat est le suivant :

```
Hello{[World]}
```

unloadMovie

Syntaxe
`unloadMovie(location);`

Arguments
location Le niveau de profondeur ou le clip cible d'où supprimer l'animation.

Description
Action ; supprime une animation du Flash Player chargée précédemment avec l'action `loadMovie`.

Lecteur
Flash 3 et versions suivantes.

Exemple
L'exemple suivant supprime l'animation principale, laissant la scène vide.

```
unloadMovie(_root);
```

L'exemple suivant supprime l'animation du niveau 15 lorsque l'utilisateur clique sur la souris.

```
on(press) {
  unloadMovie(_level15);
}
```

Voir aussi
« loadMovie » à la page 293

updateAfterEvent

Syntaxe
`updateAfterEvent(movie clip event);`

Arguments
movie clip event Vous pouvez spécifier une des valeurs suivantes comme événement de clip :

- `mouseMove` L'action est initiée chaque fois que la souris est déplacée. Utilisez les propriétés `_xmouse` et `_ymouse` pour déterminer la position actuelle de la souris.
- `mouseDown` L'action est initiée si le bouton gauche de la souris est pressé.

- `mouseUp` L'action est initiée si le bouton gauche de la souris est relâché.
- `keyDown` L'action est initiée lorsqu'une touche est enfoncée. Utilisez la méthode `Key.getCode` pour extraire les informations concernant la dernière touche enfoncée.
- `keyUp` L'action est initiée lorsqu'une touche est relâchée. Utilisez la méthode `key.getCode` pour extraire les informations concernant la dernière touche enfoncée.

Description
Action ; met à jour l'affichage (indépendant des images par seconde définies pour l'animation) après que l'événement de clip spécifié dans les arguments soit achevé. Cette action n'est pas répertoriée dans le panneau Actions de Flash. Utiliser `updateAfterEvent` avec des actions de glissement spécifiant les propriétés `_x` et `_y` pendant le déplacement de la souris permet aux objets de glisser uniformément sans effet saccadé à l'écran.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« onClipEvent » à la page 338

_url

Syntaxe
instancename._url

Arguments
instancename Le clip cible.

Description
Propriété (lecture seule) ; extrait l'URL du fichier SWF d'où le clip a été téléchargé.

Lecteur
Flash 4 et versions suivantes.

var

Syntaxe

```
var variableName1 [= value1] [...,variableNameN [=valueN]];
```

Arguments

variableName Le nom de la variable à déclarer.

value La valeur affectée à la variable.

Description

Action ; utilisée pour déclarer des variables locales. Si vous déclarez des variables locales au sein d'une fonction, les variables sont définies pour la fonction et expirent à la fin de l'appel de la fonction. Si les variables ne sont pas déclarées au sein d'un bloc, mais que la liste d'actions a été exécutée avec une action `call`, les variables sont locales et expirent à la fin de la liste courante. Si les variables ne sont pas déclarées au sein d'un bloc et que la liste courante d'actions n'a pas été exécutée avec une action call, les variables ne sont pas locales.

Lecteur

Flash 5 et versions suivantes.

_visible

Syntaxe

```
instancename._visible
instancename._visible = Booléen;
```

Arguments

Booléen Entrez une valeur *true* ou *false* pour spécifier si le clip est visible.

Description

Propriété ; détermine si le clip spécifié dans l'argument *instancename* est ou n'est pas visible. Les clips qui ne sont pas visibles (propriété définie sur `false`) sont désactivés. Par exemple, un bouton dans un clip dont la propriété `_visible` est définie sur `false` ne peut pas être cliqué.

Lecteur

Flash 4 et versions suivantes.

void

Syntaxe

```
void (expression);
```

Arguments

expression Une expression de n'importe quelle valeur.

Description

Opérateur ; un opérateur unaire qui supprime la valeur *expression* et renvoie une valeur indéfinie. L'opérateur `void` est souvent utilisé pour évaluer une URL afin de tester les effets de côté sans afficher l'expression évaluée dans la fenêtre de navigation. L'opérateur `void` est également utilisé dans les comparaisons avec l'opérateur `==` pour tester des valeurs indéfinies.

Lecteur

Flash 5 et versions suivantes.

while

Syntaxe

```
while(condition) {
statement(s);
}
```

Arguments

condition L'instruction qui est réévaluée chaque fois que l'action `while` est exécutée. Si l'instruction évalue `true`, l'expression dans *statement(s)* est lancée.

statement(s) L'expression à lancer si la condition évalue `true`.

Description

Action ; lance une instruction ou une série d'instructions de manière répétée dans une boucle aussi longtemps que l'argument de condition est `true`. Á la fin de chaque action `while`, Flash redémarre la boucle en retestant la condition. Si la condition est `false` ou égale à 0, Flash passe à la première instruction après l'action `while`.

La boucle est souvent utilisée pour effectuer une action pendant qu'une variable compteur est inférieure à une valeur spécifiée. Á la fin de chaque boucle, le compteur est incrémenté jusqu'à ce que le seuil de la valeur soit atteint, la *condition* n'est plus `true` et la boucle se termine.

Lecteur

Flash 4 et versions suivantes.

Exemple

Cet exemple duplique cinq clips sur la scène, chacun avec une position *x* et *y* générée aléatoirement, ainsi que les propriétés `xscale`, `yscale` et `_alpha` pour obtenir un effet éclaté. La variable `foo` est initialisée avec la valeur 0. L'argument *`condition`* est défini de sorte que la boucle `while` soit lancée cinq fois ou aussi longtemps que la variable `foo` est inférieure à 5. Au sein de la boucle `while`, un clip est dupliqué et `setProperty` est utilisé pour ajuster les diverses propriétés du clip dupliqué. La dernière instruction de la boucle incrémente `foo` de sorte que lorsque la valeur atteindra 5, l'argument *`condition`* évalue `false` et la boucle ne sera plus exécutée.

```
on(release) {
   foo = 0;
   while(foo < 5) {
        duplicateMovieClip("/flower", "mc" + foo, foo);
        setProperty("mc" + foo, _x, random(275));
        setProperty("mc" + foo, _y, random(275));
        setProperty("mc" + foo, _alpha, random(275));
        setProperty("mc" + foo, _xscale, random(200));
        setProperty("mc" + foo, _yscale, random(200));
        foo = foo + 1;
    }
}
```

Voir aussi

« do...while » à la page 261
« continue » à la page 241

_width

Syntaxe

`instancename``._width`
`instancename``._width =`*`value`*`;`

Arguments

`value` La largeur d'une animation en pixels.

`instancename` Le nom d'occurrence du clip pour lequel la propriété `_width` est définie ou extraite.

Description

Propriété ; définit la largeur de l'animation. Dans les versions précédentes de Flash, `_height` et `_width` étaient des propriétés en lecture seule ; dans Flash 5, elles peuvent être définies aussi bien qu'extraites.

Lecteur

Flash 4 en tant que propriété en lecture seule. Dans Flash 5 et les versions suivantes, cette propriété peut être définie aussi bien qu'extraite.

Exemple

L'exemple de code suivant définit les propriétés height et width d'un clip lorsque l'utilisateur clique sur la souris.

```
onclipEvent(mouseDown) {
   _width=200;
   _height=200;
}
```

Voir aussi

« _height » à la page 277

with

x209E6 | IDS_ACTIONHELP_WITH, instruction with

Syntaxe

```
with (object) {
statement(s);
}
```

Arguments

object L'occurrence d'un objet ActionScript ou d'un clip.

statement(s) Une action ou un groupe d'actions entre accolades.

Description

Action ; change temporairement la portée (ou le chemin cible) utilisée pour évaluer les expressions et les actions dans *statement(s)*. Une fois l'action `with` exécutée, la chaîne de portée est rétablie dans son état d'origine.

L'*object* devient le contexte dans lequel les propriétés, les variables et les fonctions sont lues. Par exemple, si *object* est `myArray` et si deux des propriétés spécifiées sont `length` et `concat`, ces propriétés sont automatiquement lues comme `myArray.length` et `myArray.concat`. Dans un autre exemple, si *object* est `state.california`, tout se passe comme si toutes les actions et instructions au sein de l'action `with` étaient appelées de l'intérieur de l'occurrence `california`.

Pour trouver la valeur d'un identifiant dans *statement(s)*, ActionScript démarre au début de la chaîne de portée spécifiée par *object* et recherche l'identifiant à chaque niveau de la chaîne de portée, dans un ordre spécifique.

La chaîne de portée utilisée par l'action `with` pour traduire les identifiants commence par le premier élément de la liste suivante et continue jusqu'au dernier, ainsi :

- *`object`* référencé par l'action `with` la plus interne
- *`object`* référencé par l'action `with` la plus externe
- Un objet d'activation (Un objet temporaire créé automatiquement lorsqu'une fonction est appelée et contient les variables locales appelées dans la fonction.)
- Le clip contenant le script en cours d'exécution
- Un objet global (un objet prédéfini comme Math ou Chaîne)

Dans Flash 5, l'action `with` remplace l'action désapprouvée `tellTarget`. Vous êtes encouragés à utiliser `with` à la place de `tellTarget` car il s'agit d'une extension standard ActionScript au standard ECMA-262. La différence principale entre les actions `with` et `tellTarget` réside dans le fait que `with` prend comme argument une référence à un clip ou à un autre objet, tandis que `tellTarget` prend une chaîne de chemin cible identifiant un clip et ne peut pas être utilisée pour cibler des objets.

Pour définir une variable dans une action `with`, la variable doit avoir été déclarée à l'extérieur de l'action `with` ou vous devez entrer le chemin complet vers le scénario dans lequel vous souhaitez que la variable existe. Si vous définissez une variable dans une action `with` sans l'avoir déclarée, l'action `with` cherchera la valeur selon la chaîne de portée. Si la variable n'existe pas déjà, la nouvelle valeur sera définie dans le scénario d'où l'action `with` a été appelée.

Exemple

L'exemple suivant définit les propriétés *x* et *y* de l'occurrence `someOtherMovieClip` puis indique à `someOtherMovieClip` d'aller à l'image 3 et d'arrêter :

```
with (someOtherMovieClip) {
  _x = 50;
  _y = 100;
  gotoAndStop(3);
}
```

L'extrait de code suivant montre comment vous auriez pu écrire le code précédent sans utiliser l'action `with`.

```
someOtherMovieClip._x = 50;
someOtherMovieClip._y = 100;
someOtherMovieClip.gotoAndStop(3);
```

Ce code pourrait également être écrit avec une action `tellTarget`.

```
tellTarget ("someOtherMovieClip") {
  _x = 50;
  _y = 100;
  gotoAndStop(3);
}
```

L'action `with` est utile pour accéder simultanément à plusieurs éléments dans une liste de chaînes de portée. Dans l'exemple suivant, l'objet intégré `Math` est placé au début de la chaîne de portée. Définir `Math` comme objet par défaut traduit les identifiants `cos`, `sin` et `PI` en `Math.cos`, `Math.sin` et `Math.PI`, respectivement. Les identifiants `a`, `x`, `y` et `r` ne sont pas des méthodes ni des propriétés de l'objet `Math`, mais puisqu'elles existent dans la portée d'activation d'objet de la fonction `polar`, elles se traduisent en variables locales correspondantes.

```
function polar(r){
  var a, x, y
  with (Math) {
    a = PI * r * r
    x = r * cos(PI)
    y = r * sin(PI/2)
}
trace("area = " +a)
trace("x = " + x)
trace("y = " + y)
}
```

Vous pouvez utiliser des actions `with` imbriquées pour accéder à des informations dans plusieurs portées. Dans l'exemple suivant, l'occurrence `fresno` et l'occurrence `salinas` sont les enfants de l'occurrence `california`. L'instruction définit les valeurs `_alpha` de `fresno` et `salinas` sans changer la valeur `_alpha` de `california`.

```
with (california){
   with (fresno){
      _alpha = 20;
   }
   with (salinas){
      _alpha = 40;
   }
}
```

Voir aussi

« tellTarget » à la page 380

_x

Syntaxe

instancename._x
instancename._x = *integer*

Arguments

integer La coordonnée locale *x* de l'animation.

instancename Le nom d'une occurrence de clip.

Description

Propriété ; définit la coordonnée *x* de l'animation en relation avec les coordonnées locales du clip parent. Si un clip se trouve dans le scénario principal, alors son système de coordonnées réfère au coin supérieur gauche de la scène sous la forme (0, 0). Si le clip se trouve dans un autre clip qui a subi des transformations, le clip est dans le système de coordonnées locales du clip imbriquant. Ainsi, pour un clip ayant pivoté de 90º dans le sens inverse des aiguilles d'une montre, les enfants de ce clip hériteront d'un système de coordonnées qui a pivoté de 90º dans le sens inverse des aiguilles d'une montre. Les coordonnées du clip réfère à la position du point d'enregistrement.

Lecteur

Flash 3 et versions suivantes.

Voir aussi

« _y » à la page 419
« _xscale » à la page 418

XML (objet)

Utilisez les méthodes et les propriétés de l'objet XML pour charger, analyser, envoyer, construire et manipuler des arborescences de documents XML.

Vous devez utiliser le constructeur `new XML()` pour créer une occurrence de l'objet XML avant d'appeler les méthodes de l'objet XML.

XML est supporté par Flash 5 et les versions suivantes de Flash Player.

Tableau des méthodes de l'objet XML

Méthode	Description
`appendChild`	Ajoute un nœud à la fin de la liste des enfants de l'objet spécifiée.
`cloneNode`	Clone le nœud spécifié et clone récursivement tous les enfants (optionnel).
`createElement`	Crée un nouvel élément XML.
`createTextNode`	Crée un nouveau nœud texte XML.
`hasChildNodes`	Renvoie `true` si le nœud spécifié possède des nœuds enfants ; sinon, renvoie `false`.
`insertBefore`	Insère un nœud devant un nœud existant dans la liste des enfants du nœud spécifiée.
`load`	Charge un document (spécifié par l'objet XML) depuis une URL.
`onLoad`	Une fonction de rappel pour load et sendAndLoad.
`parseXML`	Analyse un document XML dans l'arborescence de l'objet XML spécifiée.
`removeNode`	Supprime le nœud spécifié de son parent.
`send`	Envoie l'objet XML spécifié à une URL.
`sendAndLoad`	Envoie l'objet XML spécifié à une URL et charge la réponse du serveur dans un autre objet XML.
`toString`	Convertit le nœud spécifié et les enfants en texte XML.

Tableau des propriétés de l'objet XML

Propriété	Description
`docTypeDecl`	Définit et renvoie les informations relatives à une déclaration DOCTYPE de document.
`firstChild`	Référence le premier enfant de la liste pour le nœud spécifié.
`lastChild`	Référence le dernier enfant de la liste pour le nœud spécifié.
`loaded`	Vérifie si l'objet XML spécifié a été chargé.
`nextSibling`	Référence le frère suivant dans la liste des enfants du nœud parent.
`nodeName`	Renvoie le nom de balise d'un élément XML.

Propriété	Description
`nodeType`	Renvoie le type du nœud spécifié (élément XML ou nœud texte).
`nodeValue`	Renvoie le texte du nœud spécifié si le nœud est un nœud texte.
`parentNode`	Référence le nœud parent du nœud spécifié.
`previousSibling`	Référence le frère précédent dans la liste des enfants du nœud parent.
`status`	Renvoie un code d'état numérique indiquant le succès ou l'échec d'une opération d'analyse d'un document XML.
`xmlDecl`	Définit et renvoie les informations relatives à une déclaration document d'un document.

Tableau des collections de l'objet XML

Méthode	Description
`attributes`	Renvoie un tableau associatif contenant tous les attributs du nœud spécifié.
`childNodes`	Renvoie un tableau contenant les références aux nœuds enfants du nœud spécifié.

Constructeur de l'objet XML

Syntaxe

```
new XML();
new XML(source);
```

Arguments

`source` Le document analysé pour créer le nouvel objet XML.

Description

Constructeur ; crée un nouvel objet XML. Vous devez utiliser la méthode constructeur pour créer une occurrence de l'objet XML avant d'appeler les méthodes de l'objet XML.

La première syntaxe crée un nouvel objet XML vide.

La seconde syntaxe construit un nouvel objet XML en analysant le document XML spécifié dans l'argument *`source`* et remplit l'objet XML nouvellement créé avec l'arborescence de documents XML résultante.

Remarque : Les méthodes `createElement` et `createTextnode` sont les méthodes constructeur pour la création des éléments et des nœuds texte dans une arborescence de documents XML.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant crée un nouvel objet XML vide.

```
myXML = new XML();
```

Voir aussi
« XML.createTextNode » à la page 398
« XML.createElement » à la page 398

XML.appendChild

Syntaxe
myXML.appendChild(*childNode*);

Arguments
childNode Le nœud enfant à ajouter dans la liste des enfants de l'objet XML spécifiée.

Description
Méthode ; ajoute le noeud enfant spécifié à la fin de la liste des enfants de l'objet XML. Le nœud enfant ajouté est placé dans la structure arborescente une fois supprimé de son nœud parent existant (s'il y en a un).

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant clone le dernier nœud de `doc1` et l'ajoute à la fin de `doc2`.

```
doc1 = new XML(src1);
doc2 = new XML();
node = doc1.lastChild.cloneNode(true);
doc2.appendChild(node);
```

XML.attributes

Syntaxe
myXML.attributes;

Arguments
Aucun.

Description
Collection (lecture-écriture) ; renvoie un tableau associatif contenant tous les attributs de l'objet XML spécifié.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant écrit les noms des attributs XML dans la fenêtre Résultat.

```
str = "<mytag name=\"Val\"> intem </mytag>";
doc = new XML(str);
y = doc.firstChild.attributes.name;
  (trace) y;
doc.firstChild.attributes.order = "first";
z = doc.firstChild.attributes.order
  trace(z);
```

La séquence suivante est écrite dans la fenêtre Résultat :

```
Val
First
```

XML.childNodes

Syntaxe
myXML.childNodes;

Arguments
Aucun.

Description
Collection (lecture seule) ; renvoie un tableau des enfants de l'objet XML spécifié. Chaque élément du tableau est une référence à un objet XML qui représente un nœud enfant. Cette propriété est en lecture seule et ne peut pas être utilisée pour manipuler des nœuds enfants. Utilisez les méthodes `appendChild`, `insertBefore` et `removeNode` pour manipuler les nœuds enfants.

Cette collection est indéfinie pour les nœuds texte (nodeType == 3).

Lecteur
Flash 5 et versions suivantes.

XML.cloneNode

Syntaxe
myXML.cloneNode(*deep*);

Arguments
deep Valeur booléenne spécifiant si les enfants de l'objet XML spécifié sont clonés récursivement.

Description
Méthode ; construit et renvoie un nouveau nœud XML possédant les mêmes type, nom, valeur et attributs que l'objet XML spécifié. Si *deep* est défini sur `true`, tous les nœuds enfants sont récursivement clonés, obtenant ainsi une copie exacte de l'arborescence de documents de l'objet original.

Lecteur
Flash 5 et versions suivantes.

XML.createElement

Syntaxe
```
myXML.createElement(name);
```

Arguments
name Le nom de balise de l'élément XML en création.

Description
Méthode ; crée un nouvel élément XML portant le nom spécifié dans l'argument. Le nouvel élément n'a, au départ, ni parent, ni enfants. La méthode renvoie une référence à l'objet XML nouvellement créé représentant l'élément. Cette méthode et `createTextNode` sont des méthodes constructeur pour créer les nœuds d'un objet XML.

Lecteur
Flash 5 et versions suivantes.

XML.createTextNode

Syntaxe
```
myXML.createTextNode(text);
```

Arguments
text Le texte utilisé pour créer un nouveau nœud texte.

Description
Méthode ; crée un nouveau nœud texte XML avec le texte spécifié. Le nouveau nœud n'a, au départ, pas de parent et les nœuds texte ne peuvent pas avoir d'enfants. Cette méthode renvoie une référence à l'objet XML représentant le nouveau nœud texte. Cette méthode et `createElement` sont les méthodes constructeur pour créer les nœuds d'un objet XML.

Lecteur
Flash 5 et versions suivantes.

XML.docTypeDecl

Syntaxe

myXML.XMLdocTypeDecl;

Arguments

Aucun.

Description

Propriété ; définit et renvoie les informations relatives à la déclaration DOCTYPE d'un document. Après l'analyse du texte XML en objet XML, la propriété `XML.docTypeDecl` est définie sur le texte de la déclaration DOCTYPE du document XML. Par exemple, `<!DOCTYPE greeting SYSTEM "hello.dtd">`. Cette propriété est définie en utilisant une représentation chaîne de la déclaration DOCTYPE, et non un objet XML noeud.

Le programme d'analyse XML d'ActionScript n'est pas un programme de validation. La déclaration DOCTYPE est lue par le programme d'analyse et stockée dans la propriété `docTypeDecl`, mais aucune validation DTD n'est effectuée.

Si aucune déclaration DOCTYPE n'est rencontrée pendant une opération d'analyse, `XML.docTypeDecl` est définie sur indéfini. `XML.toString` sort le contenu de `XML.docTypeDecl` immédiatement après la déclaration XML stockée dans `XML.xmlDecl` et avant tout autre texte dans l'objet XML. Si `XML.docTypeDecl` est indéfini, aucune déclaration DOCTYPE n'est sortie.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `XML.docTypeDecl` pour définir la déclaration DOCTYPE d'un objet XML.

```
myXML.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

Voir aussi

« XML.toString » à la page 409

« XML.xmlDecl » à la page 409

XML.firstChild

Syntaxe

myXML.firstChild;

Arguments

Aucun.

Description
Propriété (lecture seule) ; évalue l'objet XML spécifié et référence le premier enfant de la liste des enfants du nœud parent. Cette propriété est `null` si le nœud n'a pas d'enfant. Cette propriété est indéfinie si le nœud est un nœud texte. Cette propriété est en lecture seule et ne peut pas être utilisée pour manipuler les nœuds enfants ; utilisez les méthodes `appendChild`, `insertBefore` et `removeNode` pour manipuler les nœuds enfants.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.appendChild » à la page 396

« XML.insertBefore » à la page 400

« XML.removeNode » à la page 407

XML.haschildNodes

Syntaxe
```
myXML.hasChildNodes();
```

Arguments
Aucun.

Description
Méthode ; évalue l'objet XML spécifié et renvoie `true` s'il y a des nœuds enfants ; sinon, renvoie `false`.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant utilise les informations de l'objet XML dans une fonction utilisateur définie.

```
if (rootNode.hasChildNodes()) {
  myfunc (rootNode.firstChild);
}
```

XML.insertBefore

Syntaxe
```
myXML.insertBefore(childNode, beforeNode);
```

Arguments
childNode Le nœud à insérer.

beforeNode Le nœud avant le point d'insertion de *childNode*.

Description
Méthode ; insère un nouveau nœud enfant dans la liste des enfants de l'objet XML, avant `beforeNode`.

Lecteur
Flash 5 et versions suivantes.

XML.lastChild

Syntaxe
```
myXML.lastChild;
```

Arguments
Aucun.

Description
Propriété (lecture seule) ; évalue l'objet XML et référence le dernier enfant de la liste des enfants du nœud parent. Cette méthode renvoie `null` si le nœud ne possède pas d'enfants. Cette propriété est en lecture seule et ne peut pas être utilisée pour manipuler des nœuds enfants ; utilisez les méthodes `appendChild`, `insertBefore` et `removeNode` pour manipuler les nœuds enfants.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.appendChild » à la page 396

« XML.insertBefore » à la page 400

« XML.removeNode » à la page 407

XML.load

Syntaxe
```
myXML.load(url);
```

Arguments
url L'URL où se trouve le document à charger. L'URL doit se trouver dans le même sous-domaine que l'URL où se trouve actuellement l'animation.

Description

Méthode ; charge un document XML depuis l'URL spécifiée et remplace le contenu de l'objet XML spécifié par les données XML téléchargées. Le processus de chargement est asynchrone ; il ne se termine pas immédiatement après l'exécution de la méthode `load`. Lorsque `load` est exécutée, la propriété `loaded` de l'objet XML est définie sur `false`. Lorsque le téléchargement des données XML est terminé, la propriété `loaded` est définie sur `true` et la méthode `onLoad` est invoquée. Les données XML ne sont pas analysées avant le complet téléchargement. Si l'objet XML contenait auparavant des arborescences XML, elles sont supprimées.

Vous pouvez spécifier votre propre fonction de rappel à la place de la méthode `onLoad`.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant est une simple utilisation de `XML.load`.

```
doc = new XML();
doc.load ("theFile.xml");
```

Voir aussi

« XML.onLoad » à la page 404
« XML.loaded » à la page 402

XML.loaded

Syntaxe

myXML`.loaded;`

Arguments

Aucun.

Description

Propriété (lecture seule) ; détermine si le processus de chargement du document initié par l'appel `XML.load` est achevé. Si le processus s'est achevé avec succès, la méthode renvoie `true` ; sinon, elle renvoie `false`.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant utilise `XML.loaded` dans un script simple.

```
if (doc.loaded) {
  gotoAndPlay(4)
}
```

XML.nextSibling

Syntaxe
myXML.nextSibling;

Arguments
Aucun.

Description
Propriété (lecture seule) ; évalue l'objet XML et référence le frère suivant dans la liste des enfants du nœud parent. Cette méthode renvoie `null` si le nœud ne possède pas un nœud frère suivant. Cette propriété est en lecture seule et ne peut pas être utilisée pour manipuler les nœuds enfants. Utilisez les méthodes `appendChild`, `insertBefore` et `removeNode` pour manipuler les nœuds enfants.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.appendChild » à la page 396
« XML.insertBefore » à la page 400
« XML.removeNode » à la page 407

XML.nodeName

Syntaxe
myXML.nodeName;

Arguments
Aucun.

Description
Propriété ; prend ou renvoie le nom du nœud de l'objet XML. Si l'objet XML est un élément XML (`nodeType` == 1), `nodeName` est le nom de la balise représentant le nœud dans le ficher XML. Par exemple, `TITLE` est le nodeName d'une balise HTML `TITLE`. Si l'objet XML est un nœud texte (`nodeType` == 3), le nodeName est `null`.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.nodeType » à la page 404

XML.nodeType

Syntaxe
myXML.nodeType;

Arguments
Aucun.

Description
Propriété (lecture seule) ; prend ou renvoie une valeur `nodeType`, avec 1 pour un élément XML et 3 pour un noeud texte.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.nodeValue » à la page 404

XML.nodeValue

Syntaxe
myXML.nodeValue;

Arguments
Aucun.

Description
Propriété ; renvoie la valeur de nœud de l'objet XML. Si l'objet XML est un nœud texte, le type de nœud est 3 et `nodeValue` est le texte du noeud. Si l'objet XML est un élément XML, l'argument `nodeValue` est `null` et est en lecture seule.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.nodeType » à la page 404

XML.onLoad

Syntaxe
myXML.onLoad(*success*);

Arguments
success Une valeur booléenne indiquant si l'objet XML a été chargé avec succès avec une opération `XML.load` ou `XML.sendAndLoad`.

Description
Méthode ; invoquée par le Flash Player lorsqu'un document XML est reçu depuis le serveur. Si le document XML est reçu avec succès, l'argument *success* est `true`. Si le document n'a pas été reçu, ou si une erreur est survenue lors de la réception de la réponse du serveur, l'argument *success* est `false`. L'implémentation par défaut de cette méthode n'est pas active. Pour avoir priorité sur l'implémentation par défaut, vous devez affecter une fonction contenant vos propres actions.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant crée une animation Flash simple pour une simple application de vitrine de commerce électronique. Nous utilisons la méthode `sendAndLoad` pour transmettre un élément XML contenant le nom et le mot de passe de l'utilisateur et nous installons un gestionnaire `onLoad` pour traiter la réponse du serveur.

```
var myLoginReply = new XML();
myLoginReply.onLoad = myOnLoad;
myXML.sendAndLoad("http://www.samplestore.com/login.cgi",
                  myLoginReply);
function myOnLoad(success) {
  if (success) {
    if (e.firstChild.nodeName == "LOGINREPLY" &&
        e.firstChild.attributes.status == "OK") {
      gotoAndPlay("loggedIn")
    } else {
      gotoAndStop("loginFailed")
    }
  } else {
    gotoAndStop("connectionFailed")
  }
}
```

Voir aussi
« function » à la page 271
« XML.load » à la page 401
« XML.sendAndLoad » à la page 407

XML.parentNode

Syntaxe
myXML.parentNode;

Arguments
Aucun.

Description
Propriété (lecture seule) ; référence le nœud parent de l'objet XML spécifié ou renvoie `null` si le nœud n'a pas de parent. Cette propriété est en lecture seule et ne peut pas être utilisée pour manipuler les nœuds enfants ; utilisez les méthodes `appendChild`, `insertBefore` et `removeNode` pour manipuler les enfants.

Lecteur
Flash 5 et versions suivantes.

XML.parseXML

Syntaxe
myXML`.parseXML(`*source*`);`

Arguments
source Le texte XML à analyser et à transmettre à l'objet `XML` spécifié.

Description
Méthode ; analyse le texte XML spécifié dans l'argument *source* et remplit l'objet XML spécifié avec l'arborescence XML résultante. Toutes les arborescences existantes dans l'objet XML sont supprimées.

Lecteur
Flash 5 et versions suivantes.

XML.previousSibling

Syntaxe
myXML`.previousSibling;`

Arguments
Aucun.

Description
Propriété (lecture seule) ; évalue l'objet XML et référence le frère précédent dans la liste des enfants du nœud parent. Renvoie `null` si le nœud ne possède pas de nœud frère précédent. Cette propriété est en lecture seule et ne peut pas être utilisée pour manipuler des nœuds enfants ; utilisez les méthodes `appendChild`, `insertBefore` et `removeNode` pour manipuler les nœuds enfants.

Lecteur
Flash 5 et versions suivantes.

XML.removeNode

Syntaxe
```
myXML.removeNode();
```

Arguments
Aucun.

Description
Méthode ; supprime l'objet XML spécifié de son parent.

Lecteur
Flash 5 et versions suivantes.

XML.send

Syntaxe
```
myXML.send(url);
myXML.send(url, window);
```

Arguments
url L'URL de destination pour l'objet XML spécifié.

window La fenêtre de navigation qui affichera les données renvoyées par le serveur : `_self` spécifie l'image courante dans la fenêtre courante, `_blank` spécifie une nouvelle fenêtre, `_parent` spécifie le parent de l'image courante et `_top` spécifie l'image de haut niveau dans la fenêtre courante.

Description
Méthode ; code l'objet XML spécifié en un document XML et l'envoie à l'URL spécifiée en utilisant la méthode `POST`.

Lecteur
Flash 5 et versions suivantes.

XML.sendAndLoad

Syntaxe
```
myXML.sendAndLoad(url,targetXMLobject);
```

Arguments
url L'URL de destination pour l'objet XML spécifié. L'URL doit se trouver dans le même sous-domaine que l'URL d'où l'animation a été téléchargée.

targetXMLobject Un objet XML créé avec la méthode constructeur XML et qui recevra les informations renvoyées depuis le serveur.

Description
Méthode ; code l'objet XML spécifié en un document XML, l'envoie à l'URL spécifiée en utilisant la méthode `POST`, télécharge la réponse du serveur, puis la charge dans *`targetXMLobject`* spécifié dans les arguments. La réponse du serveur est chargée de la même manière que celle utilisée dans la méthode `load`.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XML.load » à la page 401

XML.status

Syntaxe
`myXML``.status;`

Arguments
Aucun.

Description
Propriété ; définit et renvoie automatiquement une valeur numérique indiquant si un document XML a été analysé avec succès dans un objet XML. La liste suivante répertorie les codes numériques d'état et décrit chacun d'entre eux.

- 0 Aucune erreur ; analyse achevée avec succès.
- -2 Une section CDATA n'a pas été achevée correctement.
- -3 La déclaration XML n'a pas été achevée correctement.
- -4 La déclaration DOCTYPE n'a pas été achevée correctement.
- -5 Un commentaire n'a pas été achevé correctement.
- -6 Un élément XML a été mal formé.
- -7 Mémoire disponible insuffisante.
- -8 Une valeur d'attribut n'a pas été achevée correctement.
- -9 Une balise de début n'a pas coïncidé avec une balise de fin.
- -10 Une balise de fin a été rencontrée sans balise de début correspondante.

Lecteur
Flash 5 et versions suivantes.

XML.toString

Syntaxe
myXML.toString();

Arguments
Aucun.

Description
Méthode ; évalue l'objet XML spécifié, construit une représentation textuelle de la structure XML en incluant le nœud, les enfants et les attributs et renvoie le résultat sous la forme d'une chaîne.

Pour les objets XML de haut niveau (ceux construits avec le constructeur), XML.toString sort la déclaration XML du document (stockée dans XML.xmlDecl), suivie de la déclaration DOCTYPE du document (stockée dans XML.docTypeDecl), suivie de la représentation textuelle de tous les nœuds XML de l'objet. La déclaration XML n'est pas sortie si XML.xmlDecl est indéfini. La déclaration DOCTYPE n'est pas sortie si XML.docTypeDecl est indéfini.

Lecteur
Flash 5 et versions suivantes.

Exemple
Le code suivant est un exemple de la méthode XML.toString.

```
node = new XML("<h1>test</h1>");
trace(node.toString());
sends
<H1>test</H1>
to the output window
```

Voir aussi
« XML.xmlDecl » à la page 409
« XML.docTypeDecl » à la page 399

XML.xmlDecl

Syntaxe
myXML.xmlDecl;

Arguments
Aucun.

Description
Propriété ; définit et renvoie les informations relatives à la déclaration XML d'un document. Après l'analyse du document XML dans un objet XML, cette propriété est définie en utilisant le texte de la déclaration XML du document. Cette propriété est définie en utilisant une représentation chaîne de la déclaration XML et non un objet nœud XML. Si aucune déclaration XML n'a été rencontrée durant une opération d'analyse, la propriété est définie sur indéfini. `XML.toString` sort le contenu de `XML.xmlDecl` avant tout autre texte dans l'objet XML. Si `XML.xmlDecl` contient le type `undefined`, aucune déclaration XML n'est sortie.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant utilise `XML.xmlDecl` pour définir la déclaration XML d'un document pour un objet XML.

```
myXML.xmlDecl = "<?xml version=\"1.0\" ?>";
```

Voir aussi
« XML.toString » à la page 409
« XML.docTypeDecl » à la page 399

XMLSocket (objet)

L'objet XMLSocket implémente les sockets client qui permettent à l'ordinateur de lancer le Flash Player pour communiquer avec un ordinateur serveur identifié par une adresse IP ou un nom de domaine.

Utilisation de l'objet XMLSocket

Pour utiliser l'objet XMLSocket, le serveur doit lancer un démon qui comprend le protocole utilisé par l'objet XMLSocket. Le protocole est le suivant :

- Les messages XML sont envoyés à travers une connexion socket TCP/IP continue en duplex intégral.
- Chaque message XML est un document XML complet, terminé par un octet zéro.
- Un nombre illimité de messages XML peut être envoyé et reçu à travers une seule connexion XMLSocket.

L'objet XMLSocket est utile pour les applications serveur-client nécessitant un faible délai (par exemple, des systèmes de dialogue en ligne en temps réel). Une solution de dialogue en ligne traditionnelle basée HTTP interroge fréquemment le serveur et télécharge les nouveaux messages à l'aide d'une requête HTTP. À l'inverse, une solution de dialogue en ligne XMLSocket maintient une connexion ouverte avec le serveur, ce qui permet au serveur d'envoyer immédiatement les messages entrants sans que le client en fasse la requête.

Paramétrer un serveur pour communiquer avec l'objet XMLSocket peut être un véritable défi. Si votre application ne nécessite pas une interactivité en temps réel, utilisez l'action `loadVariables` ou les connexions serveur XML basée HTTP de Flash (`XML.load`, `XML.sendAndLoad`, `XML.send`), à la place de l'objet XMLSocket.

Pour utiliser les méthodes de l'objet XMLSocket, vous devez d'abord utiliser le constructeur, `new XMLSocket`, pour créer un nouvel objet XMLSocket.

XMLSocket et sécurité

Étant donné que l'objet XMLSocket établit et maintient une connexion ouverte avec le serveur, les restrictions suivantes ont été placées sur l'objet XMLSocket pour des raisons de sécurité.

- La méthode `XMLSocket.connect` ne peut être connectée qu'à des numéros de port TCP supérieurs ou égaux à 1024. En conséquence de cette restriction, les démons du serveur qui communiquent avec l'objet XMLSocket doivent également être affectés à des numéros de port supérieurs ou égaux à 1024. Les numéros de port inférieurs à 1024 sont souvent utilisés par des services de système (par exemple, FTP, Telnet et HTTP), interdisant ainsi l'objet XMLSocket pour ces ports. La restriction du numéro de port limite les possibilités d'accès inappropriés et abusifs à ces ressources.
- La méthode `XMLSocket.connect` ne peut se connecter qu'à des ordinateurs du même sous-domaine que celui où réside le fichier (l'animation) SWF. Cette restriction ne s'applique pas aux animations tournant sur un disque local. (Cette restriction est identique aux règles de sécurité de `loadVariables`, `XML.sendAndLoad` et `XML.load`.)

Tableau des méthodes de l'objet XMLSocket

Méthode	Description
`close`	Ferme une connexion socket ouverte.
`connect`	Établit une connexion vers le serveur spécifié.
`onClose`	Une fonction de rappel invoquée lorsqu'une connexion XMLSocket est fermée.
`onConnect`	Une fonction de rappel invoquée lorsqu'une connexion XMLSocket est établie.
`onXML`	Une fonction de rappel invoquée lorsqu'un objet XML provient du serveur.
`send`	Envoie un objet XML au serveur.

Constructeur de l'objet XMLSocket

Syntaxe

```
new XMLSocket();
```

Arguments

Aucun.

Description

Constructeur ; crée un nouvel objet XMLSocket. L'objet XMLSocket n'est pas connecté à un serveur au début. Vous devez appeler la méthode `XMLSocket.connect` pour connecter l'objet à un serveur.

Lecteur

Flash 5 et versions suivantes.

Exemple

```
myXMLSocket = new XMLSocket();
```

Voir aussi

« XMLSocket.connect » à la page 413

XMLSocket.close

Syntaxe
myXMLSocket.close();

Arguments
Aucun.

Description
Méthode ; ferme la connexion spécifiée par l'objet XMLSocket.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« XMLSocket.connect » à la page 413

XMLSocket.connect

Syntaxe
myXMLSocket.connect(*host, port*);

Arguments
host Un nom de domaine DNS pleinement qualifié ou une adresse IP au format *aaa.bbb.ccc.ddd*. Vous pouvez également spécifier null pour vous connecter au serveur hôte où se trouve l'animation.

port Le numéro de port TCP de l'hôte utilisé pour établir une connexion. Le numéro de port doit être supérieur ou égal à 1024.

Description
Méthode ; établit une connexion avec l'hôte Internet spécifié en utilisant le port TCP spécifié (supérieur ou égal à 1024) et renvoie true ou false selon que la connexion a été établie ou non avec succès. Si vous ne connaissez pas le numéro du port de votre machine hôte Internet, contactez votre administrateur réseau. Si le module externe Netscape de Flash ou si le contrôle ActiveX est utilisé, l'hôte spécifié dans l'argument doit posséder le même sous-domaine que l'hôte d'où l'animation a été téléchargée.

Si vous spécifiez null pour l'argument *host*, l'hôte contacté sera l'hôte où l'appel d'animation XMLSocket.connect se trouve. Par exemple, si l'animation a été téléchargée à l'adresse http://www.yoursite.com, spécifier null pour l'argument *host* revient à entrer l'adresse IP www.yoursite.com.

Si XMLSocket.connect renvoie la valeur true, la première étape du processus de connexion est un succès ; plus tard, la méthode XMLSocket.onConnect est invoquée pour déterminer si la connexion finale a réussi ou échoué. Si XMLSocket.connect renvoie false, la connexion n'a pas pu être établie.

Lecteur
Flash 5 et versions suivantes.

Exemple
L'exemple suivant utilise `XMLSocket.connect` pour se connecter à l'hôte où se trouve l'animation et utilise `trace` pour afficher la valeur renvoyée indiquant le succès ou l'échec de la connexion.

```
function myOnConnect(success) {
  if (success) {
    trace ("Connection succeeded!")
  } else {
    trace ("Connection failed!")
  }
}
socket = new XMLSocket()
socket.onConnect = myOnConnect
if (!socket.connect(null, 2000)) {
  trace ("Connection failed!")
}
```

Voir aussi
« function » à la page 271
« XMLSocket.onConnect » à la page 415

XMLSocket.onClose

Syntaxe
myXMLSocket.onClose();

Arguments
Aucun.

Description
Méthode ; une fonction de rappel qui n'est invoquée que lorsqu'une connexion ouverte est fermée par le serveur. L'implémentation par défaut de cette méthode n'effectue aucune action. Pour avoir la priorité sur l'implémentation par défaut, vous devez affecter une fonction contenant vos propres actions.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« function » à la page 271
« XMLSocket.onConnect » à la page 415

XMLSocket.onConnect

Syntaxe

myXMLSocket.onConnect(*success*);

Arguments

success Une valeur booléenne indiquant si une connexion socket a été établie avec succès (true ou false).

Description

Méthode ; une fonction de rappel invoquée par le Flash Player lorsqu'une requête de connexion initiée par la méthode XMLSocket.connect a réussi ou échoué. Si la connexion a réussi, l'argument *success* est true ; sinon, l'argument *success* est false.

L'implémentation par défaut de cette méthode n'effectue aucune action. Pour avoir la priorité sur l'implémentation par défaut, vous devez affecter une fonction contenant vos propres actions.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant illustre le processus de spécification d'une fonction de remplacement pour la méthode onConnect dans une simple application de dialogue en ligne.

La fonction contrôle les écrans vers lesquels les utilisateurs sont dirigés selon que la connexion a été établie avec succès ou non. Si la connexion est établie avec succès, les utilisateurs sont dirigés vers l'écran de dialogue principal de l'image nommée startChat. Si la connexion est un échec, les utilisateurs sont dirigés vers un écran d'informations de dépannage dans l'image nommée connectionFailed.

```
function myOnConnect(success) {
  if (success) {
    gotoAndPlay("startChat")
  } else {
    gotoAndStop("connectionFailed")
  }
}
```

Après avoir créé l'objet XMLSocket à l'aide de la méthode constructeur, le script installe la méthode onConnect en utilisant l'opérateur d'affectation :

```
socket = new XMLSocket()
socket.onConnect = myOnConnect
```

Pour finir, la connexion est initiée. Si `connect` renvoie `false`, l'animation est envoyée directement dans l'image nommée `connectionFailed` et `onConnect` n'est jamais invoquée. Si `connect` renvoie `true`, l'animation va dans une image appelée `waitForConnection`, qui est l'écran « Please wait ». L'animation reste sur l'image `waitForConnection` jusqu'à ce que le gestionnaire `onConnect` soit invoqué, ce qui arrive à un moment donné dans le futur, selon le délai d'attente du réseau.

```
if (!socket.connect(null, 2000)) {
  gotoAndStop("connectionFailed")
} else {
  gotoAndStop("waitForConnection")
}
```

Voir aussi

« XMLSocket.connect » à la page 413
« function » à la page 271

XMLSocket.onXML

Syntaxe

myXMLSocket.onXML(*object*);

Argument

object Une occurrence de l'objet XML contenant un document XML analysé reçu d'un serveur.

Description

Méthode ; une fonction de rappel invoquée par le Flash Player lorsque l'objet XML spécifié contenant un document XML arrive par une connexion XMLSocket ouverte. Une connexion XMLSocket peut être utilisée pour transférer un nombre illimité de documents XML entre le client et le serveur. Chaque document est terminé par un octet zéro. Lorsque le Flash Player reçoit l'octet zéro, il analyse tous les XML reçus depuis l'octet zéro précédent ou depuis que la connexion a été établie s'il s'agit du premier message reçu. Chaque lot de XML analysé est traité comme un seul document XML et transféré à la méthode `onXML`.

L'implémentation par défaut de cette méthode n'effectue aucune action. Pour avoir la priorité sur l'implémentation par défaut, vous devez affecter une fonction contenant vos propres actions.

Lecteur

Flash 5 et versions suivantes.

Exemple

La fonction suivante prend la priorité sur l'implémentation par défaut de la méthode `onXML` dans une simple application de dialogue en ligne. La fonction `myOnXML` instruit l'application de dialogue en ligne pour reconnaître un seul élément XML, `MESSAGE`, dans le format suivant :

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.
```

Le gestionnaire `onXML` doit d'abord être installé dans l'objet XMLSocket ainsi :

```
socket.onXML = myOnXML;
```

La fonction `displayMessage` est considérée comme étant une fonction définie utilisateur qui affiche le message reçu à l'utilisateur.

```
function myOnXML(doc) {
  var e = doc.firstChild;
  if (e != null && e.nodeName == "MESSAGE") {
    displayMessage(e.attributes.user, e.attributes.text);
  }
}
```

Voir aussi

« function » à la page 271

XMLSocket.send

Syntaxe

myXMLSocket.send(*object*);

Arguments

object Un objet XML ou d'autres données à transmettre au serveur.

Description

Méthode ; convertit l'objet XML ou les données spécifiés dans l'argument *object* en chaîne et la transmet au serveur, suivie d'un octet zéro. Si *object* est un objet XML, la chaîne est la représentation textuelle XML de l'objet XML. L'opération send est asynchrone ; elle revient immédiatement, mais les données peuvent être transmises plus tard. La méthode `XMLSocket.send` ne renvoie pas de valeur indiquant si les données ont été transmises avec succès.

Si l'objet *myXMLSocket* n'est pas connecté au serveur (avec `XMLSocket.connect`), l'opération `XMLSocket.send` sera un échec.

Lecteur

Flash 5 et versions suivantes.

Exemple

L'exemple suivant illustre comment spécifier un nom d'utilisateur et un mot de passe pour envoyer l'objet XML `myXML` au serveur.

```
var myXML = new XML();
var myLogin = myXML.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
myXML.appendChild(myLogin);
myXMLSocket.send(myXML);
```

Voir aussi

« XMLSocket.connect » à la page 413

_xmouse

Syntaxe

instancename._xmouse

Arguments

instancename Le nom d'une occurrence de clip.

Description

Propriété (lecture seule) ; renvoie la coordonnée *x* de la position de la souris.

Lecteur

Flash 5 et versions suivantes.

Voir aussi

« Mouse (objet) » à la page 311
« _ymouse » à la page 420

_xscale

Syntaxe

instancename._xscale
instancename._xscale = *percentage*;

Arguments

percentage Une valeur de pourcentage spécifiant le pourcentage de redimensionnement horizontal de l'animation. La valeur par défaut est 100.

instancename Le nom d'une occurrence de clip.

Description
Propriété ; détermine l'échelle horizontale (*percentage*) du clip selon l'application du point d'enregistrement du clip. Le point d'enregistrement par défaut est (0,0).

Redimensionner le système de coordonnées locales affecte les définitions des propriétés _x et _y, définies en pixels. Par exemple, si le clip parent est dimensionné à 50%, définir la propriété _x bouge un objet de l'animation de la moitié du nombre de pixels qu'il aurait si l'animation était à 100%.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« _xscale » à la page 418

_y

Syntaxe
```
instancename._y
instancename._y = integer;
```

Arguments
integer La coordonnée locale *y* du clip.

instancename Le nom d'une occurrence de clip.

Description
Propriété ; définit la coordonnée *y* d'une animation en fonction des coordonnées locales du clip parent. Si un clip se trouve dans le scénario principal, alors son système de coordonnées fait référence au coin supérieur gauche de la scène, sous la forme (0, 0). Si le clip se trouve dans un clip qui a subi des transformations, le clip est dans le système de coordonnées locales du clip imbriquant. Ainsi, pour un clip ayant pivoté de 90º dans le sens inverse des aiguilles d'une montre, les enfants de ce clip hériteront d'un système de coordonnées qui a pivoté de 90º dans le sens inverse des aiguilles d'une montre. Les coordonnées du clip réfère à la position du point d'enregistrement.

Lecteur
Flash 3 et versions suivantes.

Voir aussi
« _yscale » à la page 420

_ymouse

Syntaxe
instancename._ymouse

Arguments
instancename Le nom d'une occurrence de clip.

Description
Propriété (lecture seule) ; indique la coordonnée *y* de la position de la souris.

Lecteur
Flash 5 et versions suivantes.

Voir aussi
« Mouse (objet) » à la page 311
« _xmouse » à la page 418

_yscale

Syntaxe
instancename._yscale
instancename._yscale = *percentage*;

Arguments
percentage Une valeur de pourcentage spécifiant le pourcentage de redimensionnement vertical de l'animation. La valeur par défaut est 100.

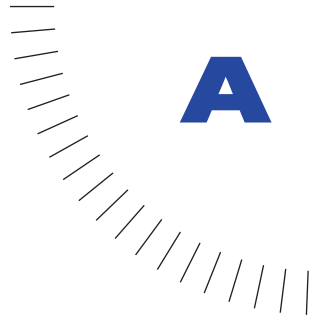
instancename Le nom d'une occurrence de clip.

Description
Propriété ; définit l'échelle verticale (*percentage*) du clip selon l'application du point d'enregistrement du clip. Le point d'enregistrement par défaut est (0,0).

Redimensionner le système de coordonnées locales affecte les définitions des propriétés _x et _y, définies en pixels. Par exemple, si le clip parent est dimensionné à 50%, définir la propriété _x bouge un objet de l'animation de la moitié du nombre de pixels qu'il aurait si l'animation était à 100%.

Lecteur
Flash 4 et versions suivantes.

Voir aussi
« _x » à la page 393
« _y » à la page 419



ANNEXE A
Priorité et associativité des opérateurs

Liste des opérateurs

Ce tableau répertorie l'ensemble des opérateurs ActionScript et leur associativité, et classe leur priorité de la plus haute à la plus faible.

Opérateur	Description	Associativité
	Priorité la plus élevée	
+	Unaire plus	Droite à gauche
-	Unaire moins	Droite à gauche
~	Complément au niveau du bit un	Droite à gauche
!	NON logique	Droite à gauche
non	NON logique (style Flash 4)	Droite à gauche
++	Post-incrémentation	Gauche à droite
--	Post-décrémentation	Gauche à droite
( )	Appel de fonction	Gauche à droite
[ ]	Élément de tableau	Gauche à droite
.	Membre de la structure	Gauche à droite
++	Pré-incrémentation	Droite à gauche
--	Pré-décrémentation	Droite à gauche
new	Allouer objet	Droite à gauche
delete	Désallouer objet	Droite à gauche

Opérateur	Description	Associativité
typeof	Type d'objet	Droite à gauche
void	Renvoie une valeur non définie	Droite à gauche
*	Multiplier	Gauche à droite
/	Diviser	Gauche à droite
%	Pourcentage	Gauche à droite
+	Additionner	Gauche à droite
add	Concaténation de chaîne (auparavant &)	Gauche à droite
-	Soustraire	Gauche à droite
«	Déplacement vers la gauche au niveau du bit	Gauche à droite
»	Déplacement vers la droite au niveau du bit	Gauche à droite
»»	Déplacement vers la droite au niveau du bit (non signé)	Gauche à droite
<	Inférieur à	Gauche à droite
<=	Inférieur ou égal à	Gauche à droite
>	Supérieur à	Gauche à droite
>=	Supérieur ou égal à	Gauche à droite
lt	Inférieur à (version chaîne)	Gauche à droite
le	Inférieur ou égal à (version chaîne)	Gauche à droite
gt	Supérieur à (version chaîne)	Gauche à droite
ge	Supérieur ou égal à (version chaîne)	Gauche à droite
==	Égal	Gauche à droite
!=	Différent	Gauche à droite
eq	Égal (version chaîne)	Gauche à droite
ne	Différent (version chaîne)	Gauche à droite
&	AND au niveau du bit	Gauche à droite
^	XOR au niveau du bit	Gauche à droite
|	OR au niveau du bit	Gauche à droite
&&	ET logique	Gauche à droite
et	ET logique (Flash 4)	Gauche à droite

Opérateur	Description	Associativité
||	OU logique	Gauche à droite
ou	OU logique (Flash 4)	Gauche à droite
?:	Conditionnel	Droite à gauche
=	Affectation	Droite à gauche
“*=, /=, %=, +=, -=, &=, |=, ^=, <<=, >>=, >>>=”	Affectation de composant	Droite à gauche
,	Évaluation multiple	Gauche à droite
	Priorité la moins élevée	

B

ANNEXE B
Touches du clavier et valeurs de code de touches

Les tableaux suivants répertorient toutes les touches d'un clavier standard et les valeurs de code des touches correspondantes utilisées pour identifier les touches dans ActionScript. Pour plus d'informations, consultez la description de l'objet Touche dans le chapitre 7, « Dictionnaire ActionScript ».

Lettres A à Z et chiffres standard de 0 à 9

Touche alphabétique ou numérique	Code de touche
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
0	48

Touche alphabétique ou numérique	Code de touche
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

Touches du clavier numérique

Touche du clavier numérique	Code de touche
0 clavier numérique	96
1 clavier numérique	97
2 clavier numérique	98
3 clavier numérique	99
4 clavier numérique	100
5 clavier numérique	101
6 clavier numérique	102
7 clavier numérique	103
8 clavier numérique	104
9 clavier numérique	105
Multiplier	106
Additionner	107
Entrée	108
Soustraire	109
Décimal	110
Diviser	111

Touches de fonction

Touche de fonction	Code de touche
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118

Touche de fonction	Code de touche
F8	119
F9	120
F10	121
F11	122
F12	123

Autres touches

Touche	Code de touche
Retour arrière	8
Tab	9
Effacer	12
Entrée	13
Maj	16
Ctrl	17
Alt	18
Verr Maj	20
Échap	27
Espace	32
Pg. Préc.	33
Pg. Suiv.	34
Fin	35
Origine	36
Gauche	37
Haut	38
Droite	39
Bas	40
Inser	45
Suppr	46
Aide	47
Verr num	144
; :	186
= +	187
- _	189
/ ?	191
` -	192

Touche	Code de touche
[{	219
\ |	220
] }	221
‘ “	222

ANNEXE C
Messages d'erreur

Le tableau suivant présente une liste de messages d'erreur renvoyés par le compilateur Flash. Chaque message est accompagné d'une explication permettant de résoudre les problèmes liés aux fichiers d'animation.

Message d'erreur	Description
Propriété <propriété> n'existe pas	Une propriété inexistante a été rencontrée. Par exemple, `x = _green` n'est pas valide, car la propriété `_green` n'existe pas.
Opérateur <opérateur> doit être suivi d'un opérande	Un opérateur sans opérande a été rencontré. Par exemple, `x = 1 +` requiert un opérande après l'opérateur +. Un opérateur est suivi d'un opérande non valide. Par exemple, `trace(1+);` est incorrect au niveau syntaxique.
Erreur de syntaxe	Ce message est émis à chaque fois que le système détecte une erreur de syntaxe non spécifique.
Nom de champ attendu après l'opérateur '.'	Vous devez spécifier un nom de champ valide en utilisant la syntaxe *`object.field`*.

<Symbole> attendu	Un symbole non valide ou inattendu a été rencontré. Par exemple, dans la syntaxe ci-dessous, le symbole `foo` n'est pas valide. Le symbole attendu est `while`.
	`do {`
	`  trace (i)`
	`} foo (i < 100)`

La liste de l'initialiseur doit se terminer par un <indicateur de fin>	Le] ou } fermant est manquant dans une liste d'initialiseurs d'objet ou de tableau.

Message d'erreur	Description
Identificateur attendu	Le système a rencontré un symbole inattendu à la place d'un identificateur. Dans l'exemple ci-dessous, 3 est un identificateur non valide. `var 3 = 4;`
La '<construction>' JavaScript n'est pas prise en charge	Une construction JavaScript non prise en charge par ActionScript a été rencontrée. Ce message apparaît si l'une des constructions JavaScript suivantes est utilisée : `void`, `switch`, `try`, `catch` ou `throw`.
Un opérateur d'affectation doit avoir une variable ou une propriété à sa gauche	Un opérateur d'affectation a été utilisé, mais la variable ou la propriété située à sa gauche n'était pas valide.
Un bloc d'instruction doit se terminer par '}'	Un groupe d'instructions a été déclaré entre accolades, mais l'accolade fermante est manquante.
Événement attendu	Un gestionnaire `On(MouseEvent)` ou `onClipEvent` a été déclaré, mais aucun événement n'a été spécifié ou le système a détecté un symbole inattendu à la place d'un événement.
Événement non valide	Le script contient un événement de souris ou de clip non valide. Pour obtenir une liste d'événements de souris et de clip valides, consultez les entrées On(MouseEvent) et OnClipEvent dans le chapitre Dictionnaire ActionScript.
Code de touche attendu	Vous devez spécifier un code de touche. Pour obtenir une liste de codes de touches, consultez l'annexe B.
Code de touche non valide	Le code de touche spécifié n'existe pas.
Caractères de fin non valides	Le script ou l'expression a été analysée correctement, mais les caractères de fin supplémentaires n'ont pas pu être analysés
Fonction non valide	Une déclaration de fonction nommée a été utilisée comme expression. Les déclarations de fonction nommées doivent être des instructions. Valide : `function sqr (x) { return x * x; }` Non valide : `function sqr (x) { return x * x; }`
Nom de fonction attendu	Le nom spécifié pour cette fonction n'est pas un nom de fonction valide.

Message d'erreur	Description
Nom de paramètre attendu	Un nom de paramètre (argument) était attendu dans une déclaration de fonction, mais un symbole inattendu a été rencontré.
'else' rencontré sans 'if'	Une instruction `else` a été rencontrée, mais sans l'instruction `if` qui la précède. Vous ne pouvez utiliser l'instruction `else` qu'avec une instruction `if`.
Type de scène erroné	L'argument de scène d'une action `gotoAndPlay`, `gotoAndStop` ou `ifFrameLoaded` était de type erroné. L'argument de scène doit être une constante de chaîne.
Erreur interne	Une erreur interne s'est produite dans le compilateur ActionScript. Envoyez le fichier FLA qui a généré cette erreur à Macromedia avec des instructions détaillées permettant de reproduire le message.
Chiffres hexadécimaux attendus après 0x	La séquence 0x a été rencontrée, mais elle n'était pas suivie de chiffres hexadécimaux valides.
Erreur lors de l'ouverture du fichier #include	Une erreur s'est produite lors de l'ouverture d'un fichier inclus dans la directive `include`. L'erreur est intervenue parce que le fichier était absent ou en raison d'une erreur disque.
Directive #include incorrecte	Une directive `include` n'a pas été rédigée correctement. Une directive `include` doit avoir la syntaxe suivante : `#include "unFichier.as"`
Un commentaire multilignes n'était pas terminé	Un commentaire multilignes commençant par `/*` n'a pas de balise `*/` fermante.
Le littéral de chaîne n'est pas terminé correctement	Un littéral de chaîne commence par un guillemet ouvrant (simple ou double), mais le guillemet fermant est manquant.
Fonction <fonction> prend <nombre> paramètres	Une fonction a été appelée, mais un nombre inattendu de paramètres a été rencontré.
Nom de propriété attendu dans GetProperty	La fonction `getProperty` a été appelée, mais le deuxième argument n'était pas le nom d'une propriété de clip.
Paramètre <paramètre> ne peut pas être déclaré plusieurs fois	Un nom de paramètre est apparu plusieurs fois dans la liste des paramètres d'une déclaration de fonction. Tous les noms de paramètre doivent être attribués une seule fois.

Message d'erreur	Description
Variable ‹variable› ne peut pas être déclarée plusieurs fois	Un nom de variable est apparu plusieurs fois dans une instruction `var`. Tous les noms de variable dans une seule instruction `var` doivent être attribués une seule fois.
Les gestionnaires 'on' ne peuvent pas être imbriqués dans d'autres gestionnaires 'on'	Un gestionnaire `on` a été déclaré à l'intérieur d'un autre gestionnaire `on`. Tous les gestionnaires `on` doivent apparaître dans le niveau supérieur d'une liste d'actions.
Une instruction doit apparaître à l'intérieur d'un gestionnaire on	Dans les actions d'une instance de bouton, une instruction a été déclarée sans être délimitée par un bloc `on`. Toutes les actions d'une instance de bouton doivent apparaître à l'intérieur d'un bloc `on`.
Une instruction doit apparaître à l'intérieur d'un gestionnaire onClipEvent	Dans les actions d'une instance de clip, une instruction a été déclarée sans être délimitée par un bloc `onClipEvent`. Toutes les actions d'une instance de clip doivent apparaître à l'intérieur d'un bloc `onClipEvent`.
Les événements de souris ne sont autorisés que pour les instances de bouton	Un gestionnaire d'événement de bouton a été déclaré dans une liste d'actions d'image ou une liste d'actions d'instance de clip. Les événements de bouton ne sont autorisés que dans les listes d'actions des instances de bouton.
Les événements de clip ne sont autorisés que pour les instances de clip	Un gestionnaire d'événement de clip a été déclaré dans une liste d'actions d'image ou une liste d'actions d'instance de bouton. Les événements de clip ne sont autorisés que dans les listes d'actions des instances de clip.

INDEX

A

B

C

D

E

F

N

O

P

R

U

V

W

X