Déploiement d'applications ASP.Net

Sommaire

Introduction

Avec ASP.Net, les sites web deviennent de véritables applications, à ce titre, elles sont "déployables" comme n'importe quel autre programme. Avec Visual Studio .Net, il est possible de créer des projets de déploiement, qui vont générer des packages d'installation MSI. Dans une certaine mesure, il est possible de personnaliser ce processus. Nous allons voir, au cours de ce tutorial, les différentes façons de déployer des applications ASP.Net, de manière artisanale et simple, mais aussi de manière beaucoup plus complexe et intégrée.

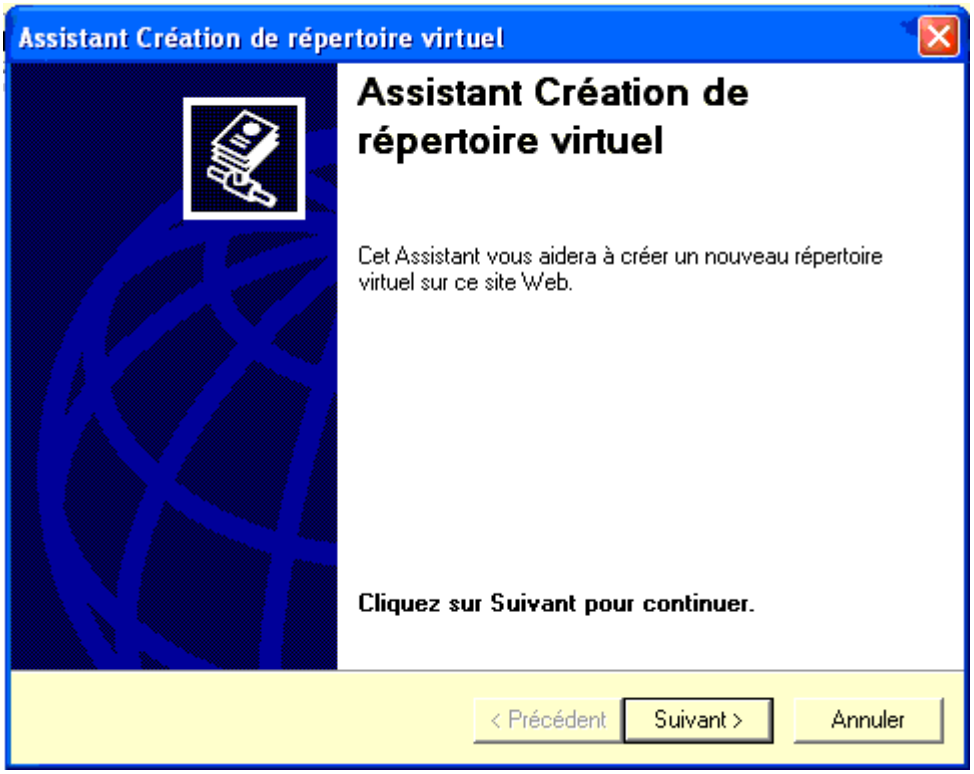
Pour réaliser ce tutoriel vous même, il est nécessaire d'avoir une configuration minimale répondant aux critères suivants :

- Visual Studio .Net 2003
- .Net Framework 1.1
- MDAC 2.6 ou supérieur
- Internet Information Services 5.0 ou supérieur
- SQL Server 2000 ou MSDE

1. Création d'un répertoire virtuel

Tout d'abord, une application ASP.Net a besoin d'un répertoire virtuel configuré sur le serveur Web pour fonctionner. La création de celui-ci sera automatique en cas de déploiement avec un package MSI, mais nous allons quand même voir comment réaliser cette opération "à la main". Voici la démarche à suivre dans IIS.

En premier, il faut aller dans la console MMC d'Internet Information Services qui se trouve dans les outils d'administration de Windows 2003. Il faut alors sélectionner votre serveur, puis le site web sur lequel sera publié votre application. Ensuite, cliquer avec le bouton droit sur ce même site, choisir "Nouveau", puis "Répertoire virtuel"; s'ouvre alors un assistant : *L'assistant de création de répertoire virtuel* (original, non ?!).



Tout d'abord, il faut donner l'alias pour ce répertoire, c'est-à-dire le nom que va employer l'utilisateur pour y accéder, concrètement si notre alias est *mon_premier_repertoire*, alors pour accéder à l'application située sur le serveur *www.developpez.com* il faudra saisir l'adresse *http://www.developpez.com/mon_premier_repertoire*. Nous serons alors redirigés sur la page par défaut de l'application configurée dans IIS.

Ensuite, l'assistant va nous demander de renseigner le répertoire physique de notre application, c'est-à-dire le chemin du répertoire dans lequel se trouvent les fichiers de notre application. Il est peut être situé dans le répertoire wwwroot de IIS où n'importe où sur le disque dur du serveur, mais, il faut garder à l'esprit que si l'on ne met pas ce répertoire dans le *wwwroot*, il va falloir faire un travail de configuration plus important au niveau des autorisations NTFS (notamment pour le compte utilisé par défaut par IIS, et celui utilisé par ASP.Net).

Vient maintenant le temps de configurer les autorisations d'accès sur ce répertoire virtuel de la part de IIS et des clients qui s'y connectent. Au minimum il faut accorder la lecture et l'exécution, pour ce qui concerne le reste, cela dépend de votre application, gardez à l'esprit que des autorisations trop importantes peuvent nuire à la sécurité. Si vous avez besoin, par exemple, que votre application écrive des fichiers, il faut définir l'autorisation en écriture au niveau du sous répertoire concerné seulement, et cela uniquement pour le compte ASP.NET ou le(s) compte(s) concerné(s).

Cliquez maintenant sur "Terminé", votre répertoire virtuel est désormais créé, c'est à l'intérieur de celui-ci que sera placé votre application, plus exactement, vous mettrez les fichiers de votre application dans le répertoire physique et les utilisateurs eux, accèderont au répertoire virtuel.

2. Déploiement avec Xcopy

Le déploiement avec la commande Xcopy n'est ni plus ni moins qu'une simple copie des fichiers de votre application depuis l'emplacement de développement vers l'emplacement de production. Attention, cela va aller très vite ! Je présente rapidement cette technique car elle a le mérite d'exister… mais, très sincèrement, elle n'est pas d'une grande utilité, du moins face à ce que nous allons voir par la suite.

Voilà la commande à saisir pour copier votre application :

```
Xcopy x:\inetpub\wwwroot\mon_rep_de_deploiement
x:\inetpub\wwwroot\mon_rep_de_prodcution
```

En clair, la syntaxe est : "**Xcopy** *source destination*"

Pour avoir plus de détails sur la syntaxe de la commande Xcopy, je vous conseille vivement de consulter la documentation intégrée dans Windows. Pour cela, ouvrez une invite de commande et tapez : **xcopy /?**

Après avoir vu une technique de déploiement plus que basique, passons aux choses sérieuses !

3. Déploiement avec Visual Studio .Net

Précédemment, nous avons vu, que pour déployer une application ASP.Net, il n'est pas nécessaire d'employer des techniques sophistiquées, un simple Xcopy suffit. Cependant, utiliser des solutions d'installation un peu plus complexes permet d'avoir une intégration et une adaptation plus grande de notre application. Pour cela, Visual Studio .Net intègre des

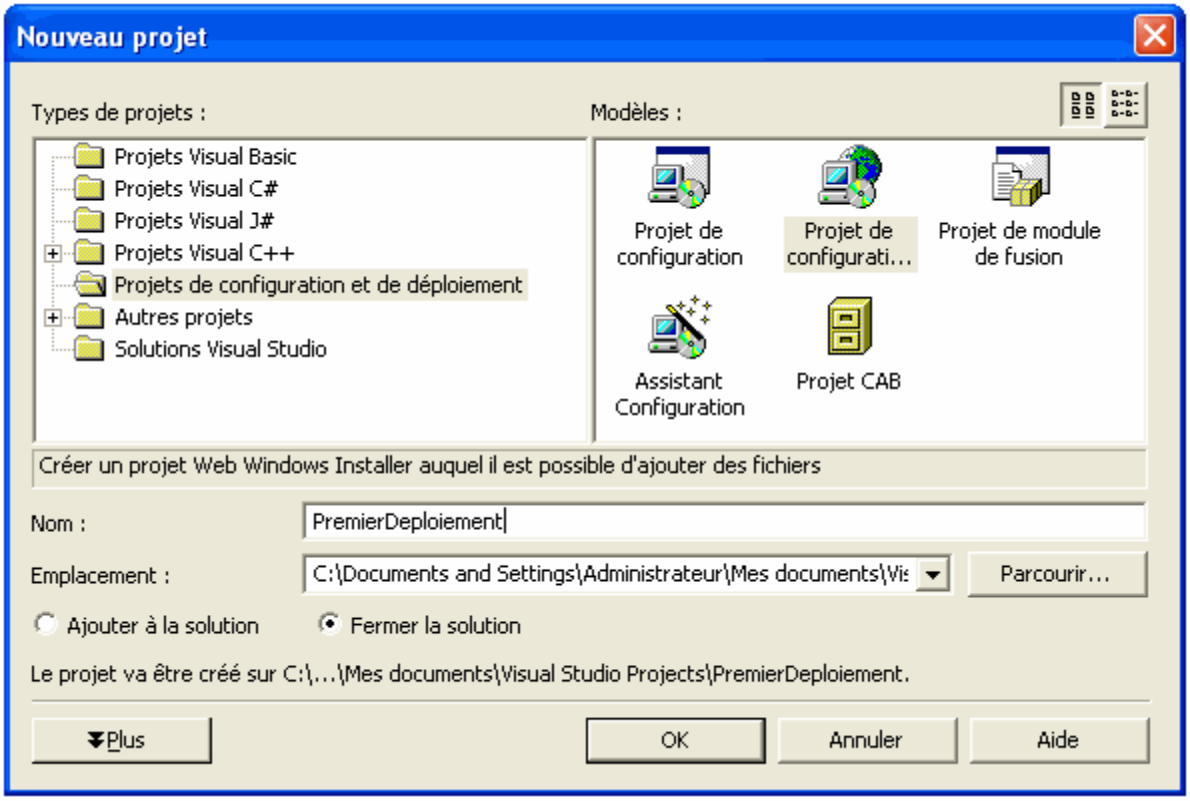
outils permettant de réaliser des packages d'installation au format MSI, pour couronner le tout, cela est d'une simplicité enfantine. Nous allons donc voir, maintenant, comment créer des packages MSI pour installer notre application.

3.1. Déploiement d'une application

Tout d'abord, je considère que votre application est terminée, c'est-à-dire que vous avez un projet terminé dans Visual Studio.Net; si tel n'est pas le cas, alors créez rapidement une application ASP.Net, peut importe ce qu'elle contient, une simple page avec une TextBox fera largement l'affaire.

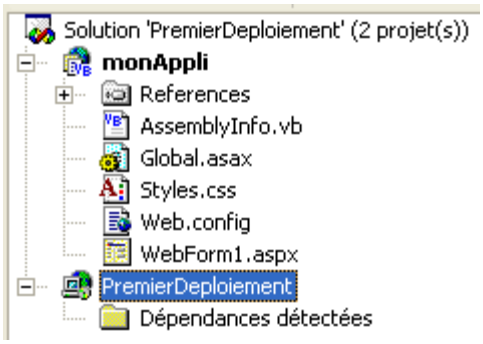
Voilà la marche à suivre :

A l'ouverture de Visual Studio, faites Fichier > Nouveau > Projet, sélectionnez alors Projets de configuration et de déploiement, puis, Projet de configuration Web. Appelez votre projet comme bon vous semble, pour moi ce sera *PremierDeploiement*. Cliquez sur Ok, le projet est alors créé par Visual Studio.



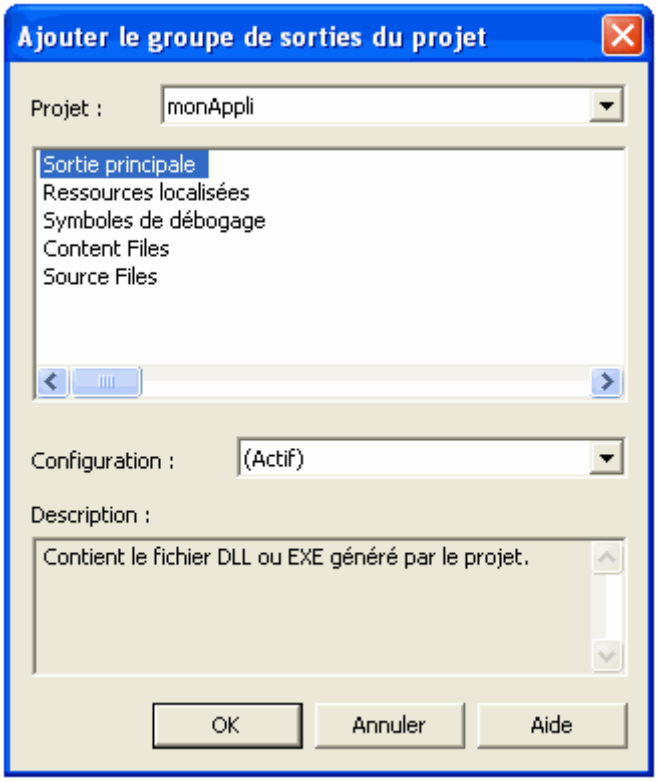
Dès lors, votre projet de déploiement est créé, il vous faut maintenant ajouter l'application que vous souhaitez déployer, pour cela, cliquez sur *Solution PremierDeploiement* dans l'explorateur de solutions, puis choisissez *ajouter un projet existant*, sélectionnez le fichier *.vbproj* de votre application et faites *ouvrir*. (Ici mon application se nomme *monAppli*).

Votre projet apparaît alors dans l'explorateur de solutions, en plus de PremierDeploiement que nous venons de créer.



Il est temps de configurer correctement notre projet de déploiement, pour cela, il faut cliquer sur le nom de votre projet dans l'explorateur de solutions, puis ensuite, aller dans la fenêtre propriété, ici vous allez compléter tout un tas de renseignements sur votre application, elles apparaîtront, pour certaines, lors de l'installation par l'utilisateur. Le nom de toutes ces propriétés étant explicite, je ne vais pas les passer en revue, sachez seulement qu'elles ne doivent pas être obligatoirement renseignées, mais cela est utile et fait plus sérieux surtout lors de l'installation. Imaginez, par exemple, que le nom s'affichant lors de l'installation soit *PremierDeploiement*, un "joli petit nom" sera quand même beaucoup mieux, idem pour les autres propriétés comme*Author* ou *Manufacturer*.

Après avoir réglé ces quelques détails, nous allons maintenant signaler au projet de déploiement quels sont les éléments à mettre dans notre package MSI et qui devront donc être installés. Pour cela, cliquez, avec le bouton droit sur *Dossier d'application Web* dans l'explorateur de fichiers (en partie centrale de l'écran), puis sur *Ajouter* et enfin *Sortie du projet*, voilà ce que vous devez voir à l'écran :



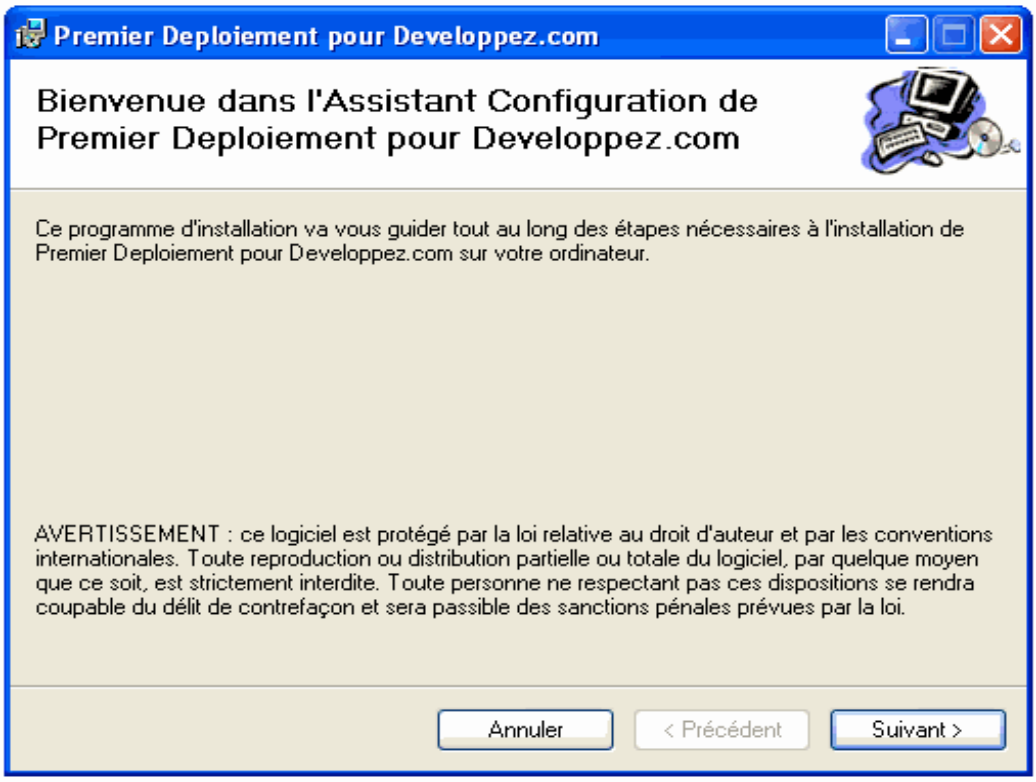
Sélectionnez alors **Sortie principale** et **Content Files** puis cliquez sur *Ok*. Vous venez donc de choisir ce qui va être copié lors de l'installation (cela comprend l'Assembly dans le répertoire*bin*, les fichiers **.aspx*, le fichier *global.asax* ainsi que le fichier *web.config*). Si vous le souhaitez, vous pouvez ajouter d'autres fichiers. Par exemple, si votre application contient des images qui se trouvent dans le répertoire \images, il vous suffit de faire :

Cliquez avec le bouton droit, sur *Dossier d'application Web*, puis *Ajouter*, *Fichier* et sélectionnez les fichiers que vous voulez copier. Si vous laissez les choses en l'état, ces fichiers seront copiés à la racine de votre application, pour les mettre dans le bon répertoire, vous pouvez accéder à la propriété *Folder* en cliquant sur les fichiers concernés dans le projet de déploiement dans*l'Explorateur de solutions*. Cela implique donc de recréer l'arborescence de répertoire de votre application dans le projet de déploiement (*Ajouter* > *Dossier Web*), mais cela uniquement pour les répertoires ne contenant pas d'élément de votre application comme des fichiers ASPX par exemple.

Dernière étape, nous allons définir la page de démarrage et le nom du dossier virtuel qui sera créée (celui-ci, néanmoins, sera modifiable par l'utilisateur lors de l'installation). Il faut cliquer sur le *Dossier d'application Web* dans l'explorateur de fichier au centre de l'écran, puis, aller dans le tableau des propriétés pour y définir au moins le **DefaultDocument** et le **VirtualDirectory**.

Votre programme d'installation est désormais quasiment prêt, il ne reste plus qu'à compiler tout cela. Tout d'abord passez en mode *Release* au lieu de *Debug* (cela se configure en haut dans la barre d'outils), puis, dans le menu *Générer*, cliquez sur *Générer la solution*. Une fois la génération terminée vous pouvez aller dans le répertoire *Release*, se trouvant dans le dossier portant le nom de votre projet de déploiement, se trouvant dans le répertoire *Visual Studio Projects*, se trouvant lui même dans *Mes Documents*. Dans ce répertoire se trouve, entre autre, le fichier MSI qui vous permettra d'installer votre application Web.

Voilà le premier écran de ce programme d'installation :



Voilà, rien de compliqué, si vous avez bien suivi toutes les étapes cela n'a pas du vous prendre plus de 2 minutes ! et encore…

Maintenant que nous savons créer un projet de déploiement de base, nous allons voir ce que nous pouvons faire pour l'améliorer un tant soit peu.

3.2. Intégration du MDAC

La quasi-totalité des applications ASP.Net sont liées à des bases de données, et sont donc appelées à utiliser le MDAC. Il s'agit du composant d'accès aux données de Microsoft. Ce dernier étant l'objet de mises à jour

relativement fréquentes, il peut être utile de le joindre à son application, pour qu'il soit mis à jour si besoin est, ainsi vous êtes sûr qu'il n'y aura pas de problème d'incompatibilité provoqué par une version trop ancienne sur le serveur où est déployée votre application. Visual Studio .Net va une fois de plus grandement vous aider, cette opération ne prendra pas plus de 2 minutes…

Voilà la démarche à suivre :

Tout d'abord, il faut télécharger le Bootstrapper qui est un plug-in pour Visual Studio .NET 2003 qui permet de distribuer le .Net Framework 1.1 et le MDAC 2.7 avec son application. Ce fichier est librement téléchargeable **ici**. Vous devez ensuite l'installer, pour cela rien de complexe, il y a juste à suivre les instructions.

Maintenant commence la véritable configuration de notre projet de déploiement :

Dans l'explorateur de solutions, sélectionnez votre projet de déploiement, puis allez dans le menu *Affichage* > *Editeur* > *Conditions de lancement*. Cliquez alors avec le bouton droit sur*Configuration requise pour l'ordinateur cible*, puis choisissez *Ajouter une condition de lancement du Registre*.

Après avoir créé notre condition, nous devons la configurer, pour cela : allez sur Recherche de *RegistryEntry1*. Dans la fenêtre Propriétés, saisissez *Software\Microsoft\DataAccess* pour *RegKey*.
Saisissez *vsdrrHKLM* dans la propriété *Root*, puis *FullInstallVer* dans la propriété *Value*. Dans *Property* tapez *MDACSEARCH*.

Sélectionnez, maintenant, *Condition1*, puis, dans la propriété Condition, entrez *MDACSEARCH>="2.7"* (ou 2.6 si, par exemple, vous souhaitez une version supérieure ou égale à 2.6, c'est évident). Vous pouvez aussi, renommer cette condition, avec un nom plus explicite, pour cela il suffit de modifier la propriété Name.

- **La propriété Message** :

Elle correspond au message qui sera affiché à l'utilisateur si une version antérieure à la 2.7 est détectée.

- **La propriété InstallUrl** :

Cela correspond à l'adresse où se situe le fichier d'installation de la version que vous souhaitez installer, par défaut le fichier se nomme mdac_typ.exe et se trouve dans le même répertoire que le fichier MSI qui est créé. Vous devez donc saisir .\mdac_typ.exe ainsi le fichier sera automatiquement lancé si l'utilisateur clique sur "Oui" quand l'"installer" lui demandera s'il souhaite mettre à jour ce composant. En cas de réponse négative, l'installation sera interrompue.

Voilà rien de plus à faire, si ce n'est de générer la solution pour que le package MSI soit modifié, et le fichier mdac_typ.exe soit ajouté.

3.3. Intégration du Framework .Net

Il est certes intéressant de pouvoir distribuer le MDAC avec notre application mais il l'est peut-être encore plus de distribuer le .Net Framework, une fois de plus Visual Studio et le Bootstrapper (qui doit être, comme dans le cas du MDAC, installé en plus de Visual Studio) viennent à notre aide. Je ne vais pas entrer dans les détails puisque la démarche est quasiment identique à celle de l'intégration du MDAC.

Démarche à suivre :

Sélectionnez votre projet de déploiement dans l'explorateur de solutions, faites un clic avec le bouton droit et sélectionnez *Propriétés*. Ensuite dans *Programme d'amorçage* sélectionnez*Programme d'amorçage de Windows Installer*, puis cliquez sur *Ok*. A la prochaine génération, le .Net Framework sera ajouté, il se trouvera dans le même dossier que le fichier MSI de votre application. Le fichier se nomme *dotnetfx.exe*.

Voilà, ainsi vous pouvez distribuer le .Net Framework avec vos applications, comme nous sommes dans le cas d'application ASP.Net, il sera assez peu utile… en effet, en général il est installé avec IIS… mais on ne sait jamais, le Framework peut toujours en être à la version 1.0 par exemple. Cela sera surtout utile avec les applications WinForm, qui sont déployées sur des machines qui sont susceptibles de ne pas avoir le Framework pré-installé.

3.4. Modification de l'interface d'installation

Les différents écrans du programme d'installation sont générés automatiquement par Visual Studio, cependant vous pouvez en ajouter d'autres, qui seront personnalisables dans une certaine mesure. Vous pouvez, par exemple, ajouter un écran contenant la licence de votre application, ou encore un autre demandant diverses informations à l'utilisateur, comme par exemple un nom ou un numéro de série.

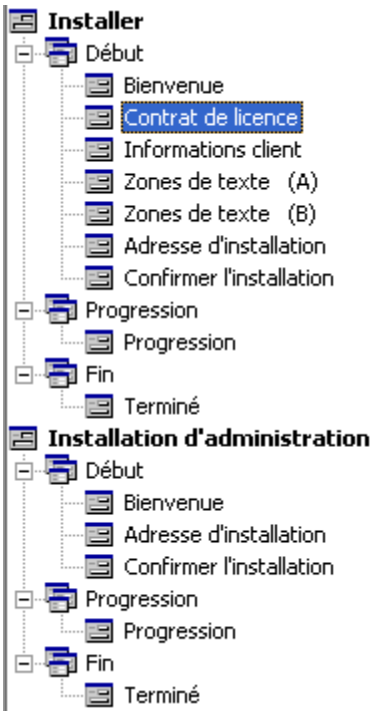
Voyons maintenant, de manière concrète, ce que l'on peut réaliser en la matière, avec Visual Studio .Net et un projet de déploiement.

3.4.1. Contrat de licence utilisateur

Ce que l'on rencontre dans l'écrasante majorité des programmes d'installation, c'est un écran présentant la licence du logiciel concerné, l'utilisateur est appelé à accepter ou à refuser cette licence, cela décide de la poursuite ou de l'interruption de l'installation. Par défaut, l'"installer", créé par Visual Studio, ne le propose pas, cependant on peut ajouter un tel écran sans difficulté, voilà les différentes étapes à suivre :

Cliquez avec le bouton droit sur votre projet de déploiement dans l'explorateur de solutions, sélectionnez *Afficher* puis *Interface utilisateur*. Un schéma apparaît alors et présente l'enchaînement des différents écrans. Sur ce schéma, on retrouve *Installer* et *Installation d'administration*, c'est le premier qui nous concerne, le deuxième reprend l'enchaînement des écrans si l'installation est réalisée à distance par exemple par l'intermédiaire d'un partage réseau (je vous invite à consulter le MSDN pour plus de détails).

Cliquez avec le bouton droit sur *Début* dans *Installer* et choisissez *Ajouter une boite de dialogue*, une fenêtre s'ouvre vous présentant les différentes types d'écrans que l'on peut ajouter. Choisissez *Contrat de licence* et cliquez sur *Ok*. Ce nouvel écran apparaît alors dans le schéma de l'*Installer*, faites un glisser/déposer avec la vignette le représentant, de telle sorte que cet écran se retrouve en deuxième position juste après l'écran de bienvenue. Votre schéma doit être conforme à l'image ci-dessous.



Cliquez sur la vignette *Contrat de licence*, dans la zone de propriétés, vous pouvez voir 3 propriétés, seules 2 nous intéressent réellement, la plus importante est LicenceFile. Il faut mettre ici le chemin vers un fichier *.rtf qui contiendra votre licence (mettez, par exemple, ce fichier à la racine de votre application). Sélectionnez-le, et cliquez sur *Ok*. Le fait que le fichier soit au format RTF permet de réaliser une mise en forme de base sur votre licence, ce qui est un plus.

L'autre propriété qui est susceptible de nous intéresser est BannerBitmap, cela pointe vers un fichier image qui sera incrusté en bannière dans l'écran d'installation, c'est donc plus du détail qu'autre chose…

Il ne vous reste plus qu'à générer la solution, voilà votre licence est intégrée dans l'"Installer", si l'utilisateur la refuse, l'installation sera tout simplement interrompue.

3.4.2. Gestion des numéros de série

Une autre personnalisation possible est l'ajout d'un écran de saisie pour un numéro de série, cela permet dans une certaine mesure de protéger votre application contre le piratage. Ainsi l'utilisateur devra entrer un numéro de série valide s'il souhaite pouvoir continuer l'installation.

Pour implémenter cela, une fois de plus, il va nous falloir moins de deux minutes. Regardez bien, ça va aller très vite ! Comme pour la licence, il faut ajouter une nouvelle boite de dialogue qui ne nomme *Informations client*, libre à vous de la placer où vous voulez dans la section début, mais à mon avis, sa place se situe juste après le contrat de licence utilisateur. En plus d'un numéro de série, il est demandé à l'utilisateur son nom et son organisation.

Cet écran contient 3 propriétés qui permettent de définir les caractéristiques de cette boite de dialogue, les voici :

- **SerialNumberTemplate** :

C'est en fait le modèle que devra prendre le numéro de série, cela crée un masque qui permettra de valider un numéro de série ou non. L'algorithme utilisé par défaut n'étant pas extrêmement complexe je vous recommande de modifier le masque de saisie proposé par défaut pour en adopter un plus long et plus complexe. Pour créer ce masque il faut utiliser des caractères spéciaux ayant une signification particulière, voilà un tableau vous permettant d'y voir plus clair.

Caractère	Signification
#	Exige un chiffre qui n'est pas inclus dans l'algorithme de validation.
%	Exige un chiffre qui est inclus dans l'algorithme de validation.
?	Exige un caractère alphanumérique qui n'est pas inclus dans l'algorithme de validation.
^	Exige un caractère majuscule ou un chiffre.
<	Tout caractère situé à gauche de ce caractère ne s'affiche pas dans la boîte de dialogue.
>	Tout caractère situé à droite de ce caractère s'affiche dans la boîte de dialogue. Obligatoire comme caractère de fin en cas d'utilisation du caractère <.

L'algorithme de validation fonctionne de la façon suivante, il additionne tous les chiffres composants le groupe concerné, il divise le résultat par 7, le reste de la division doit être 0 sinon le numéro de série est refusé. Si jamais vous restez perplexe face au fonctionnement de ce système, je vous renvoie au MSDN qui détaille un peu plus tout cela.

- **ShowOrganization** :

Cette propriété si elle est à "True" permet d'afficher une zone de texte dans laquelle l'utilisateur devra saisir le nom de son organisation, au contraire du numéro de série, le remplissage de cette zone est facultatif.

- **ShowSerialNumber** :

Par défaut, la saisie du numéro de série est désactivée, pour changer cela, il suffit de passer cette propriété à *True*. Si elle est à *False*, aucun numéro de série n'est demandé à l'utilisateur.

Vous savez désormais comment faire pour obliger l'utilisateur à saisir un numéro de série. Il ne reste plus qu'à générer la solution pour modifier en conséquence le package MSI.

Visual Studio propose une grande variété de boite de dialogue, le but ici n'est pas d'en faire une présentation exhaustive, si vous souhaitez plus d'informations je vous renvoie une fois de plus vers le MSDN. Cependant, nous allons, indirectement, découvrir d'autres boites de dialogues dans cet article.

3.5. Modification du fichier Web.config lors de l'installation

Ici, nous allons passer la vitesse supérieure, en effet nous allons modifier notre application en fonction des informations saisies par l'utilisateur. De nombreuses informations sont stockées dans le fichier de configuration (web.config), c'est le cas par exemple des chaînes de connexion des bases de données, le problème est que ces paramètres changent en fonction des utilisateurs, c'est pour ça qu'il serait très pratique que ce soit l'utilisateur lui-même qui saisisse ces informations au cours de l'installation. Nous allons donc dans cette partie apprendre à intégrer dans notre application les informations saisies à l'installation.

3.5.1. Action personnalisée

Il va nous falloir utiliser des *actions personnalisées*. Les actions personnalisées sont une fonctionnalité de Windows Installer permettant d'exécuter du code à la fin d'une installation pour effectuer différentes actions qui ne peuvent pas être réalisées au cours de l'installation. Le code à exécuter va se trouver dans une classe spécifique, il s'agît d'une classe de programme d'installation (ou Classe Installer).

Notre objectif, dans cette partie, sera de faire saisir à l'utilisateur, l'adresse IP du serveur SQL Server, le login utilisé et le mot de passe correspondant, pour ensuite modifier le fichier Web.config où ils sont sensés être stockés.

Voyons maintenant comment procéder pour arriver à un tel résultat (nous partons du principe que votre application ASP.Net est terminée, et que le projet de déploiement est créé).

Tout d'abord, il faut créer la classe d'installation, sélectionnez votre projet (l'application, et non le projet de déploiement) dans l'*explorateur de solutions*, puis il faut faire *Ajouter* > *Ajouter un nouvel élément* > *Classe Installer*, et enfin, cliquer sur *Ok*. Par défaut, la classe va être nommée *Installer1.vb*.

Il faut maintenant créer l'action personnalisée qui nous permettra de récupérer les données saisies pendant l'installation.

Allez sur votre projet de déploiement dans l'explorateur de solutions, cliquez avec le bouton droit > *Afficher* > *Action personnalisée*. La liste des actions personnalisées apparaît, pour le moment, il n'y en a aucune. Cliquez sur *Actions personnalisées*, puis faites, *Ajouter une action personnalisée*, double-cliquez sur *Dossier d'application* puis cliquez sur *Sortie principale de monAppli (Actif)*. Vous avez alors 4 actions qui apparaissent.

Nous devons maintenant configurer cette action, pour cela, allez dans la fenêtre Propriétés, dans CustomActionData tapez "*/Serveur=[EDITSERVER] /Login=[EDITLOGIN] /Password=[EDITPASSWORD]*", sans les guillemets. La signification de cette ligne est simple. Prenons le premier en exemple : *Server* représente en quelque sorte la variable récupérée, le nom entre crochets représente le nom de la zone texte où l'utilisateur va entrer l'adresse IP du serveur, ou alors son nom. Les zones textes par défaut ont bien entendu d'autres noms, il faudra par la suite leur attribuer les noms choisis ci-dessus. Vous devez modifier le CustomActionData pour les 4 nœuds, à savoir : Installer, Valider, Restaurer, Supprimer.

3.5.2. Zone de texte dans l'interface utilisateur

Vient désormais le temps d'ajouter l'écran de saisie dans notre programme d'installation, pour cela :

- Allez sur votre **Projet de déploiement**
- puis **Afficher**
- et enfin **Interface utilisateur**

- Allez alors sur **Début**
- puis **Ajouter une boite de dialogue**
- et choisissez **Zone de texte A**

Comme précédemment dans ce tutorial, déplacez la boite de dialogue dans la séquence de façon à ce qu'elle soit avant *Adresse d'installation*, sans cela vous ne pourrez pas générer votre projet.

Sélectionnez la boite de dialogue *Zone de texte A* et allez dans la fenêtre des propriétés, modifiez alors celles-ci de la façon suivante :

BannerText : Paramétrage de la base de données.
BodyText : Veuillez saisir les informations relatives à votre base de données.

Edit1Label : Nom ou adresse IP du serveur
Edit1Text :
Edit1Property : EDITSERVER
Edit1Visible : True

Edit2Label : Login de connexion
Edit2Text :
Edit2Property : EDITLOGIN
Edit2Visible : True

Edit3Label : Mot de passe
Edit3Text :
Edit3Property : EDITPASSWORD
Edit3Visible : True

Edit4Label :
Edit4Text :
Edit4Property :
Edit4Visible : False

Voilà, votre boite de dialogue est correctement configurée. Reste à coder la classe Installer. Le but du code de cette classe est de récupérer les informations saisies par l'utilisateur et de modifier le fichier web.config. Ouvrez donc le fichier Installer.vb, affichez le code. Vous pouvez voir qu'il y a déjà du code dedans. Voilà le code qu'il faut ajouter :

3.5.3. Classe de programme d'installation :

Classe d'installation

```
Public Overrides Sub Install(ByVal stateSaver As
System.Collections.IDictionary)

        MyBase.Install(stateSaver)
        ' Récupère les paramètres grâce à l'action
personnalisée.
        Dim txtServer As String =
Me.Context.Parameters.Item("Serveur")
        Dim txtUser As String =
Me.Context.Parameters.Item("Login")
        Dim txtPassword As String =
Me.Context.Parameters.Item("Password")


        'Récupère le chemin de l'assembly
        Dim Asm As System.Reflection.Assembly =
System.Reflection.Assembly.GetExecutingAssembly
        Dim chemin As String

        'Petit bricolage pour localiser le fichier
Web.config qui ne se trouve pas au même endroit que
l'assembly
        chemin = Asm.Location.Substring(0,
Asm.Location.Length - Asm.GetName.Name.Length - 8) +
"\Web.config"

        Dim FileInfo As System.IO.FileInfo = New
System.IO.FileInfo(chemin)
        If Not FileInfo.Exists Then
            Throw New InstallException("Impossible de
trouver le fichier de configuration !")
        End If

        ' Chargement du fichier de configuration.
        Dim XmlDocument As New System.Xml.XmlDocument
        XmlDocument.Load(FileInfo.FullName)
```

```
        ' Localise les noeuds cherchés et modifie les
valeurs
        Dim Node As System.Xml.XmlNode
        Dim FoundServer As Boolean = False
        Dim FoundUser As Boolean = False
        Dim FoundPassword As Boolean = False

        For Each Node In
XmlDocument.Item("configuration").Item("appSettings")
            If Node.Name = "add" Then

                If
Node.Attributes.GetNamedItem("key").Value =
"ServerName" Then

Node.Attributes.GetNamedItem("value").Value =
txtServer
                    FoundServer = True
                End If

                If
Node.Attributes.GetNamedItem("key").Value = "UserSQL"
Then

Node.Attributes.GetNamedItem("value").Value = txtUser
                    FoundUser = True
                End If

                If
Node.Attributes.GetNamedItem("key").Value =
"PasswordSQL" Then

Node.Attributes.GetNamedItem("value").Value =
txtPassword
                    FoundPassword = True
                End If

            End If
        Next Node

        If Not FoundServer Then
            Throw New InstallException("Impossible de
trouver la clé ServerName")
        End If

        If Not FoundUser Then
            Throw New InstallException("Impossible de
trouver la clé UserSQL")
        End If

        If Not FoundPassword Then
```

```
            Throw New InstallException("Impossible de
trouver la clé PasswordSQL")
        End If

        'Ecriture du fichier Web.config modifié
        XmlDocument.Save(FileInfo.FullName)


    End Sub
```

Attention : il faut avoir un certain nombre d'Imports dans cette classe, en voilà la liste:

Imports de la classe
```
Imports System.IO
Imports System.Data.SqlClient
Imports System.Reflection
Imports System.ComponentModel
Imports System.Configuration.Install
```

Comme les commentaires vous l'on expliqué, ce code permet de récupérer les valeurs saisies par l'utilisateur, puis charge le fichier Web.config et enfin, mets à jour les valeurs dans votre fichier de configuration. Voilà un extrait de mon fichier Web.config :

Extrait du fichier Web.config
```
…
<appSettings>
  <!--  Base de données -->
        <add key="ServerName" value="" />
        <add key="UserSQL" value="" />
        <add key="PasswordSQL" value="" />

</appSettings>
…
```

Il vous faudra modifier le code de la classe d'installation en fonction des noms employés pour les nœuds dans votre fichier de configuration.

3.6. Création d'une base de données lors de l'installation

Les applications ASP.Net utilisent très fréquemment une base de données, partant de ce constat, il est évident qu'en cas d'installation sur un serveur, il faudra, en plus, créer la base de données et les tables qui la compose. Avec Visual Studio, ce ne sera pas nécessairement le cas, en effet, nous allons voir comment créer automatiquement la base de données au cours de l'installation de notre application. Pour arriver à nos fins, nous allons utiliser une action personnalisée, et un fichier texte contenant un script

SQL. L'utilisateur n'aura qu'à saisir ses identifiants pour la connexion à la base de données, ainsi que le nom qu'il souhaite lui attribuer.

Nous allons nous appuyer sur ce que nous avons déjà vu précédemment. En effet, pour personnaliser le fichier de configuration, l'utilisateur doit saisir, l'adresse du serveur de base de données, ainsi que le login et le password d'un compte ayant les droits sur cette même base. Il nous suffira de récupérer ces informations pour la création de la base, inutile d'infliger à l'utilisateur une saisie de plus lors de l'installation.

Remarque : nous sommes ici dans une situation simple, dans "la vraie vie", le compte utilisé par l'application (stocké dans le fichier de configuration) n'a pas, pour des raisons évidentes de sécurité, les droits suffisants pour créer une base. Pour contourner cela, il faudra ajouter un écran de plus à notre installation pour demander à l'utilisateur de saisir un compte (login + password), ayant les autorisations adéquates. Mais cela ne nous concerne pas ici.

Passons aux choses concrètes !

Nous voulons créer une base de données automatiquement, ce qui serait intéressant serait de pouvoir choisir son nom. Voici la démarche à suivre pour cela :

Il faut configurer notre action personnalisée en conséquence, après avoir afficher nos actions (clique droit sur le projet de déploiement, puis *Afficher*, puis *Actions personnalisées*) allez dans la fenêtre Propriétés, dans CustomActionData modifiez ce que vous avez entrer précedemment "*/Serveur=[EDITSERVER] /Login=[EDITLOGIN] /Password=[EDITPASSWORD]", par " /Serveur=[EDITSERVER] /Login=[EDITLOGIN] /Password=[EDITPASSWORD] /dbname=[EDITNAME]*".

Ensuite nous devons ajouter une zone de saisie pour récupérer le nom de notre base de données à créer : clique droit sur le projet de déploiement, puis *Afficher*, puis *Interface utilisateur*, sélectionnez alors la vignette *Zone de texte A*, et dans la fenêtre des propriétés modifiez ces quelques valeurs.

Edit4Label : EDITNAME
Edit4Text : Nom que vous souhaitez donner à votre base de données
Edit4Property :
Edit4Visible : True

Pour arriver à nos fins, il ne nous faut plus que deux choses supplémentaires :

- Un script SQL pour la création de la base de données.
- Quelques lignes de code dans la classe d'installation pour exécuter le script SQL.

Pour créer le script SQL je vous renvoie à vos cours sur le langage T-SQL. Voici un exemple de ce que cela peut donner, dans un cas très basique.

Exemple de script SQL

```
CREATE TABLE [dbo].[testDVPdeploy] (
  [idclient] [varchar] (20) COLLATE French_CI_AS NOT
NULL ,
  [idsession] [varchar] (50) COLLATE French_CI_AS NOT
NULL ,
  [famille client] [varchar] (20) COLLATE French_CI_AS
NOT NULL ,
  [refarticle] [varchar] (20) COLLATE French_CI_AS NOT
NULL ,
  [prixht] [float] NOT NULL ,
  [quantite] [int] NOT NULL ,
  [valide] [varchar] (15) COLLATE French_CI_AS NOT
NULL ,
  [datecommande] [datetime] NOT NULL ,
  [prixqteht] [float] NULL ,
  [produit] [varchar] (50) COLLATE French_CI_AS NULL
) ON [PRIMARY];
```

Pour l'intégrer à votre projet dans Visual Studio, il vous suffit d'ajouter un fichier texte à la racine de votre application, et d'y copier l'intégralité de votre script, puis renommer le fichier en MaBase.sql (uniquement pour des raisons de logique et de compréhension).

Il n'est pas nécessaire d'ajouter ce fichier à votre projet de déploiement, car, par défaut, les fichiers SQL (ce n'est pas le cas des fichiers .txt) sont intégrés dans le package MSI. Mais si ce n'est pas le cas, il n'est pas ardu d'y remédier, dans l'explorateur de solutions :

- Cliquez avec le bouton droit sur votre projet de déploiement
- Cliquez sur *Ajouter*, puis sur *Fichier*
- Sélectionnez votre fichier puis cliquez sur *Ouvrir*

Et voilà, votre script SQL est intégré dans votre projet de déploiement. Passons à l'écriture du code qui va permettre d'exécuter ce script.

Remarque : rien ne vous empêche d'ajouter, en tout dernier, quelques lignes de codes pour effacer ce fichier, si vous ne voulez pas laisser le script de création à la vue de l'utilisateur. Mais bon,

cela n'est utile que pour les paranoïaques, et encore… :)

Dans votre classe d'installation, ajoutez à la suite, le code qui suit. C'est ce code qui va gérer la connexion au serveur de base de données, ainsi que l'exécution du script de création de votre base de données.

Code de création de la base de données

```
Protected Sub CreationBase(ByVal nomBase As String)

        Try
            'Création de la base de données
            ExecuteSql("master", "CREATE DATABASE " &
nomBase)

            'Exécution du script de création des
tables
            ExecuteSql(nomBase, RequeteSQL("sql.txt"))

            'Gestion des exceptions
        Catch ex As Exception

            MsgBox("Une exception à été levée : " &
ex.Message)
            Throw ex

        End Try

    End Sub

    Private Function RequeteSQL(ByVal Name As String)
As String

        Try

            'Récupère l'assembly de l'application
            Dim Asm As System.Reflection.Assembly =
System.Reflection.Assembly.GetExecutingAssembly

            'Récupère le chemin du fichier contenant
le script SQL
            Dim chemin As String =
Asm.Location.Substring(0, Asm.Location.Length -
Asm.GetName.Name.Length - 8) + "\maBase.sql"

            Dim monReader As StreamReader = New
StreamReader(chemin)

            'Lecture du fichier SQL
            Return monReader.ReadToEnd()
```

```
        Catch ex As Exception

            MsgBox("In GetSQL: " & ex.Message)
            Throw ex

        End Try

    End Function

    Private Sub ExecuteSql(ByVal DatabaseName As
String, ByVal Sql As String)

        'Récupération des informations saisies par
l'utilisateur
        Dim UserDB As String =
Me.Context.Parameters.Item("UserSQL")
        Dim PasswordDB As String =
Me.Context.Parameters.Item("PasswordSQL")
        Dim ServerDB As String =
Me.Context.Parameters.Item("ServerName")

        'Concaténation des identifiants dans la chaine
de connexion
        Dim chaine As String = "packet size=4096;user
id=" & UserDB & ";data source=" & ServerDB &
                               ";persist security
info=True;initial catalog=master;password=" &
PasswordDB & ""
        Dim SqlConnection1 = New
System.Data.SqlClient.SqlConnection
        SqlConnection1.ConnectionString = chaine


        Dim Command As New SqlClient.SqlCommand(Sql,
SqlConnection1)

        Command.Connection.Open()

Command.Connection.ChangeDatabase(DatabaseName)

        Try
            'Execution de la requete (script SQL de
création)
            Command.ExecuteNonQuery()

        Finally

            'Fermeture de la connexion
            Command.Connection.Close()

        End Try
```

```
    End Sub
```

Puis, dans **Public Overrides Sub Install(...)** il suffit d'ajouter la ligne suivante :

CreationBase(Me.Context.Parameters.Item("dbname"))

C'est cette ligne qui va lancer la création de la base de données, en appelant les méthodes nécessaires. Il ne vous reste plus qu'à générer le projet de déploiement.

Conclusion

Si vous lisez ces mots c'est que vous avez tenu bon ! Je reconnais, c'est assez long, mais il est difficile de faire plus court sans perdre du monde en route. Nous avons vu au travers de cet article qu'il est possible de réaliser des choses relativement intéressantes (et encore je suis resté au niveau le plus basique possible pour que cela ne soit pas indigeste… après c'est à vous de voir ce que vous voulez modifier pour adapter ces solutions à vos besoins), même avec Visual Studio .Net, dont le but premier n'est pas le déploiement d'applications. Certes, il n'est peut-être pas tout à fait à la hauteur, en terme de fonctionnalités, face à la concurrence. Mais tout de même, sans beaucoup d'efforts, nous sommes arrivés à intégrer nombre de choses utiles dans le déploiement d'applications.

Désormais, la modification du fichier de configuration ou la création de base de données n'ont plus de secret pour vous. De plus, avec les actions personnalisées et les classes d'installation, il est possible de faire beaucoup d'autres choses, comme par exemple la décompression d'archives, ou la création de répertoire… Dans ce domaine, votre imagination et vos talents de développeur seront certainement plus rapidement un obstacle que l'outil lui-même. J'espère que cet article sur le déploiement d'application vous a intéressé, et je vous donne rendez vous, pour de nouvelles publications prochainement.