

Création d'un template de projet personnalisé dans Visual Studio .Net

par [MORAND Louis-Guillaume](#)

Date de publication : 20/05/2005

Dernière mise à jour :

Tutoriel pas à pas expliquant le principe, le développement et l'intégration d'un template de projet dans Visual Studio .Net

Avant-propos

1 - Le template

1.1 - Introduction

1.2 - Fonctionnement

2 - Création du fichier VSDIR

2.1 - Environnement

2.2 - Création du fichier

2.3 - Personnalisation

3 - Création du fichier VSZ

4 - Personnalisation des scripts

4.1 - Création des dossiers

4.2 - Création du script

5 - Création des fichiers templates

Conclusion

Remerciements

Avant-propos

L'une des fonctionnalités les plus utilisées de Visual Studio .Net est l'utilisation de templates de fichiers sources. Que ce soit pour créer un projet winform, webform, console ou même une simple dll, vous utilisez régulièrement ces templates. Nous verrons donc au cours de cet article qu'il est possible de créer et d'utiliser nos propres templates.

Nous utiliserons ici Visual Studio .Net 2003. Certains paramètres pourront changer avec d'autres versions du même produit.

De plus, tout au long de cet article, j'utiliserai le terme anglais "*wizard*" au lieu mot français "assistant" car le mot anglais désigne spécifiquement ce type d'assistant(suivant, précédent, etc)

1 - Le template

1.1 - Introduction

Si le grand nombre de templates de Visual Studio permet de remplir la plupart des besoins des développeurs, il est possible pour de nouvelles technologies ou pour optimiser le temps de travail (projet répétitif), de créer des templates totalement personnalisés. En effet, l'extensibilité de Visual Studio est quasi infinie, elle peut être réalisée à l'aide d' add-ins, de macros, ou justement des templates.

Il existe différents types de templates que vous pourrez utiliser via l'interface de Visual Studio. Nous avons les templates de projet accessibles via le menu **Fichier > Nouveau > Projet** et les templates d'items accessibles via le menu **Projet > Ajouter un nouvel élément**.

Nous nous intéresserons ici qu'aux templates de projet, le template d'item étant une version simplifiée de ce dernier.

1.2 - Fonctionnement

La création d'un template de projet pour visual studio se fait en plusieurs étapes:

- la création d'un fichier VDIR (ou modification d'un fichier existant)
- la création d'un fichier VSZ
- la personnalisation des fichiers javascripts par défaut
- la création des fichiers templates (source)

Ce sont ces étapes que nous verrons tout du long de cet article.

2 - Création du fichier VSDIR

2.1 - Environnement

Le fichier VSDIR est un simple fichier texte avec une extension .vsdir. Il contient des informations qu'utilise Visual Studio dans les fenêtres **Nouveau Projet** et **Ajouter Item**. Ces informations sont entre autres, le nom, la description, l'icône ou même l'ordre dans lequel le template apparaîtra.

En fonction du langage dans lequel vous voudrez créer votre template, le chemin où vous placerez vos fichiers changera. Nous prendrons ici, l'exemple d'un template de projet en C# et le chemin du fichier VSDIR sera donc:

...\Microsoft Visual Studio .NET 2003\VC#\CSharpProjects

2.2 - Création du fichier

Positionnez vous donc dans le repertoire des projets C# (CSharpProjects) puis créez un fichier texte. Remplissez avec le texte suivant (**sans les sauts de ligne!!!**) et renommez le en *monTemplate.vsdir*.

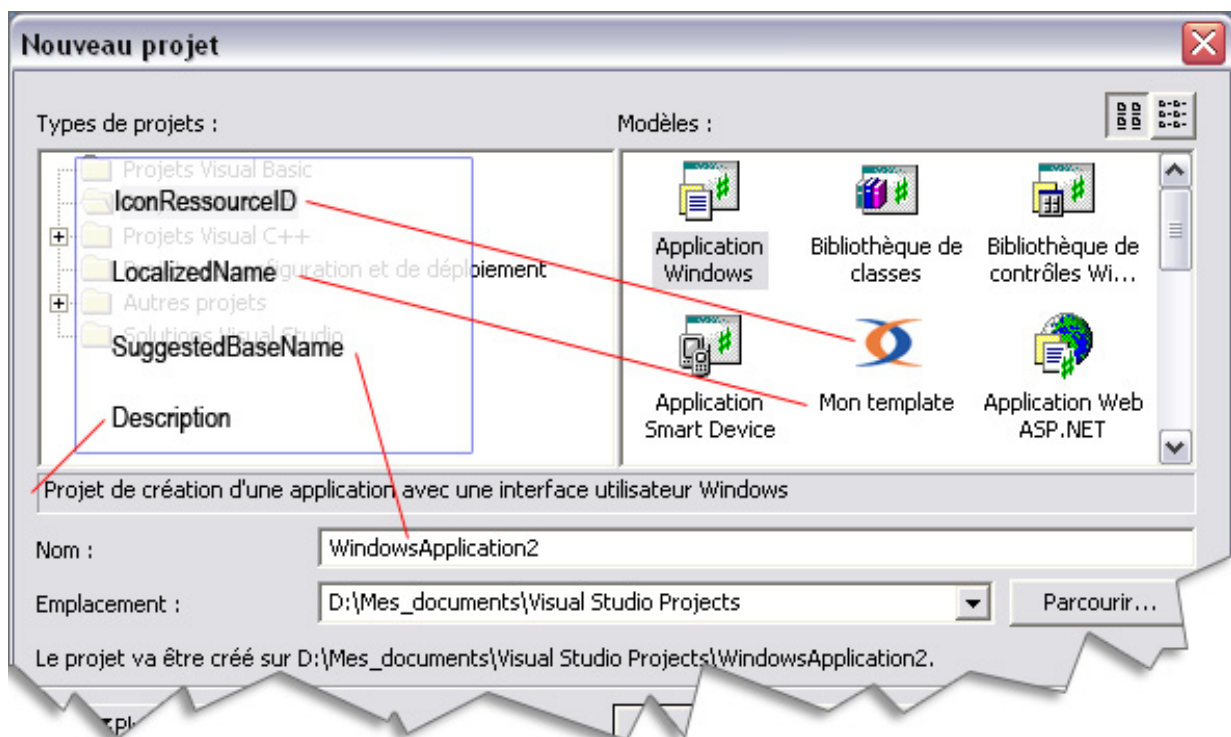
```
monTemplate.vsz | {FAE04EC1-301F-11d3-BF4B-00C04F79EFBC} |
Mon template|35|Description de mon template|
{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}|8224|0|
MonTemplateDefaultName
```

Voici le tableau descriptif des différents champs.

Numéro	Champ	Explication
1	RelPathName	Chemin du fichier .vsz pour le wizard
2	{clsidPackage}	(Optionnel) GUID représentant un produit qui contient des ressources localisées
3	LocalizedName	(Optionnel) Nom du template tel qu'il apparaîtra dans la fenêtre. Peut être soit une chaîne de caractères soit une ressource intégrée (ex: #ResID)
4	SortPriority	Priorité. Chiffre représentant l'ordre dans lequel le template apparaîtra dans la fenêtres de templates. 1 représente la première position
5	Description	Description du template
6	DLLPath or {clsidPackage}	Chemin de dll ou GUID d'un produit qui possède une .dll qui contient l'icône qui sera affichée pour le template. (utilise le IconResourceId)
7	IconResourceId	(Optionnel) #ResID de l'icône qui sera affiché dans la fenêtre des templates.
8	Flags	Différents flags (voir tableau suivant)
9	SuggestedBaseName	Le choix d'un template propose toujours un nom de projet par default.

Numéro	Champ	Explication
		Ce champ contient une chaîne de caractères ou une ressource incorporée. Si le nom de projet n'est pas unique, Visual Studio se chargera d'ajouter un chiffre derrière. Ex: monTemplate1.cs

Voici un schéma représentant le résultat final en fonction des champs du fichier VSDIR.



Voici maintenant le tableau récapitulatif des différents flags (drapeaux). Dans le fichier VSZ, vous devrez indiquer la somme des chiffres individuels de chaque flag. Par exemple, un champ *flags* égal à 9 correspondra à VSDIRFLAG_NonLocalTemplate | VSDIRFLAG_DontAddDefExtension

Nom du flag	Valeur décimale	Description
VSDIRFLAG_NonLocalTemplate	1	Utiliser une interface utilisateur non localisée et enregistrer les mécanismes
VSDIRFLAG_BlankSolution	2	Crée une solution vide. Ne crée donc pas un projet.
VSDIRFLAG_DisableBrowseButton	4	Désactive le bouton parcourir pour ce projet ou item
VSDIRFLAG_DontAddDefExtension	8	Ne pas ajouter l'extension par défaut au nom fourni pour l'item

Nom du flag	Valeur décimale	Description
VSDIRFLAG_DisableLocationField	32	Désactive le champ location pour ce projet ou item
VSDIRFLAG_DontInitNameField	4096	Ne pas initialiser le champs nom pour ce projet ou item avec un nom valide
VSDIRFLAG_DisableNameField	8192	Désactive le champ nom pour ce projet ou item

Il existe néanmoins quelques règles à suivre:

- Les champs optionnels ne contenant aucune information doivent être remplis avec 0. Note perso: s'ils sont vides, cela marche aussi
- Si aucun LocalizedName n'est proposé, Visual Studio utilisera le nom du chemin relatif
- Si aucune icône n'est proposée, Visual Studio utilisera l'icône par défaut pour l'extension de ce projet ou alors utilisera une icone vide
- Si aucun SuggestedBaseName n'est proposé, Visual Studio le remplacera par "Project".

Note: le GUID utilisé plus haut n'est pas un GUID au hasard. Il correspond au produit: Microsoft C# Code Generator for Web Discovery. Vous pouvez trouver d'autres GUID comme le générateur VB.NET dans le registre à la clé suivante: [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\7.1\Generators

2.3 - Personnalisation

Nous avons vu qu'il est possible d'utiliser les icônes déjà existantes des autres templates de Visual Studio. Nous allons maintenant voir comment utiliser nos propres icônes.

Cette étape est extrêmement simple puisqu'elle nécessite simplement de mettre une icône (32*32) dans le dossier des fichiers VSDIR et l'icône doit simplement avoir le même nom que le fichier VSDIR. Ici, nous la nommerons donc: monTemplate.ico.

Note: placer une icône dans le dossier des templates outrepassé des champs d'icônes spécifiés dans le fichier VSDIR. Cette icône est prioritaire.

3 - Création du fichier VSZ

Nous allons maintenant créer le fichier VSZ. Positionnez vous dans le même répertoire dans lequel vous avez créé votre fichier VSDIR. Créez maintenant un fichier texte et collez le texte suivant à l'intérieur:

```
VSWIZARD 7.0
Wizard=VsWizard.VsWizardEngine.7.1
Param="WIZARD_NAME = MonTemplate"
Param="WIZARD_UI = FALSE"
Param="PROJECT_TYPE = CSPROJ"
```

Renommez alors ce fichier en monTemplate.vsz. Ce nom doit correspondre au champ RelPathName que vous avez indiqué dans votre fichier VSDIR.

Voyons maintenant la signification de chaque champ :

Champ	Description
VSWIZARD	Identifie l'interface de wizard que le wizard attend pour être implémenté.
Wizard	Le ProgID du wizard
Param	Vous pouvez ajouter un grand nombre de paramètres. Un seul paramètre par ligne.

Une fois ce fichier créé, vous pouvez lancer Visual Studio et créer un nouveau projet. Votre template devrait alors être visible. Vous pouvez alors cliquer dessus et lancer votre projet; vous obtiendrez alors une erreur vous signifiant que les fichiers scripts sont manquants. Nous allons donc les créer.

4 - Personnalisation des scripts

Dans le chapitre précédent, nous avons configuré notre fichier pour utiliser le wizard par défaut. Ce dernier cherche les fichiers scripts dans le dossier **...Microsoft Visual Studio .NET 2003\VC#VC#Wizards**.

4.1 - Création des dossiers

Le wizard utilise alors le nom du template pour trouver le dossier correspondant. Créez alors le dossier:

...Microsoft Visual Studio .NET 2003\VC#VC#Wizards\monTemplate

Ce nom de dossier correspond au paramètre WIZARD_NAME du fichier VSZ.

Créez maintenant le dossier contenant les scripts:

...Microsoft Visual Studio .NET 2003\VC#VC#Wizards\monTemplate\Scripts\1033

Ainsi que le dossier qui contiendra les fichiers templates:

...Microsoft Visual Studio .NET 2003\VC#VC#Wizards\monTemplate\Templates\1033

4.2 - Création du script

Nous allons maintenant créer le fichier javascript. Son fonctionnement est plus difficile que les étapes précédentes. Contentez-vous pour le moment de copier le code javascript ci-dessous:

```
Default.js
function OnFinish(selProj, selObj)
{
    var oldSuppressUIValue = true;
    try
    {
        oldSuppressUIValue = dte.SuppressUI;
        var strProjectPath = wizard.FindSymbol("PROJECT_PATH");
        var strProjectName = wizard.FindSymbol("PROJECT_NAME");
        var strSafeProjectName = CreateSafeName(strProjectName);
        wizard.AddSymbol("SAFE_PROJECT_NAME", strSafeProjectName);

        var bEmptyProject = 0; //wizard.FindSymbol("EMPTY_PROJECT");

        var proj = CreateCSharpProject(strProjectName, strProjectPath, "default.csproj");

        var InfFile = CreateInfFile();
        if (!bEmptyProject && proj)
        {
            AddReferencesForClass(proj);
            AddFilesToCSharpProject(proj, strProjectName, strProjectPath, InfFile,
false);
        }
    }
    catch(e)
    {
        if( e.description.length > 0 )
            SetErrorInfo(e);
        return e.number;
    }
    finally

```

```

Default.js
{
    dte.SuppressUI = oldSuppressUIValue;
    if( InfFile )
        InfFile.Delete();
}

function GetCSharpTargetName(strName, strProjectName)
{
    var strTarget = strName;

    switch (strName)
    {
        case "fichier1.cs":
            strTarget = "classel.cs";
            break;
        case "TODO.txt":
            strTarget = "TODOlist.txt";
            break;
    }
    return strTarget;
}

function DoOpenFile(strName)
{
    var bOpen = false;

    switch (strName)
    {
        case "classel.cs":
            bOpen = true;
            break;
        case "fichier2.cs":
            bOpen = true;
            break;
        case "TODOlist.txt":
            bOpen = false;
            break;
    }
    return bOpen;
}

function SetFileProperties(oFileItem, strFileName)
{
    if(strFileName == "classel.cs" || strFileName == "fichier2.cs")
    {
        oFileItem.Properties("SubType").Value = "Code";
    }
}

```

Ce fichier est une copie modifiée du fichier script par défaut du template de création d'un projet console.

Je pense que même pour des personnes maîtrisant bien le javascript, ce bout de code et son fonctionnement n'est pas forcément clair. Voyons donc comment fonctionne ce fichier javascript:

1- La méthode OnFinish est appelée lorsque l'utilisateur clique sur le bouton "Finir" de la fenêtre de dialogue de création d'un nouveau projet. La méthode récupère alors différentes variables "d'environnement" passées par le wizard via Visual Studio. Nous récupérons par exemple le nom et le chemin du projet.

Toujours à l'intérieur de cette méthode, nous appelons la méthode CreateCSharpProject. Cette dernière fait partie des fonctions de base qui sont contenues dans le fichier ...\\Microsoft Visual Studio .NET 2003\\VC#\\VC#Wizards\\1036\\common.js

Cette méthode exécute différentes actions comme supprimer la solution déjà ouverte (si option cochée par l'utilisateur), puis création du projet à partir du nom donné dans la fenêtre de création de projet.

Il utilise alors le fichier default.csproj pour créer le projet C#. Il est possible d'utiliser votre propre fichier csproj et de le personnaliser selon vos besoins.

Enfin, la méthode AddFilesToCSharpProject, lit le fichier Templates.inf et ajoute chaque fichier qu'il contient au projet.

2- C'est au tour de la méthode GetCSharpTargetName. Cette méthode détermine le nom final du fichier dans l'application. Note, il suffit sinon, de bien nommer les fichiers templates dès le départ :)

3- La méthode DoOpenFile est alors appelée. Cette méthode boucle sur tous les items du projet et ouvre chacun d'entre eux ou non. Dans notre exemple: à la création du projet, les fichiers *classe1.cs* et *fichier2.cs* sont ouverts par l'éditeur de Visual Studio tandis que *TODOList.txt* ne l'est pas.

4- Enfin la méthode SetFileProperties est appelée. Cette dernière définit le type de fichier. Ici, nos deux fichiers sont des fichiers de code mais nous aurions pu avoir des fichier de type "Form" par exemple.

Passons à la création propre des templates de fichier sources

5 - Création des fichiers templates

Il peut être très intéressant de s'inspirer des fichiers templates existants afin de créer des bons fichiers templates. Néanmoins, les fichiers templates sont des fichiers sources "basiques" qui contiennent certains "champs" dynamiques dont les valeurs seront indiquées par Visual Studio. Nous prendrons ici un cas relativement simple, mais sachez qu'il est possible de faire des solutions complètement générées à partir de templates et contenant un grand nombre de classes et/ou d'objets.

Nous créons ici un projet C# console contenant deux fichiers sources ainsi qu'un fichier servant de to-do list rudimentaire.

Ces fichiers seront les fichiers: fichier1.cs, fichier2.cs et TODO.txt

Commençons par créer le fichier Templates.inf (attention à la casse!) que vous placerez dans le dossier \Templates\1033

Puis, remplissez-le avec trois lignes contenant le nom de chaque fichier à inclure:

Templates.inf

```
fichier1.cs
fichier2.cs
TODO.txt
```

Créez alors les fichiers fichier1.cs et fichier2.cs dans lesquels vous copierez les codes suivants.

fichier1.cs

```
namespace [!output SAFE_NAMESPACE_NAME] {
    using System;
    ///
    /// Nom du projet: [!output SAFE_NAMESPACE_NAME].
    /// Classe : [!output SAFE_CLASS_NAME]
    ///
    class [!output SAFE_CLASS_NAME] {
        //
        // Methode principale
        //
        static void Main(string[] args) {
            fichier2 monObjet = new fichier2();
        }
    }
}
```

fichier2.cs

```
namespace [!output SAFE_NAMESPACE_NAME] {
    using System;
    ///
    /// Nom du projet: [!output SAFE_NAMESPACE_NAME].
    /// Classe : [!output SAFE_CLASS_NAME]
    ///
    class [!output SAFE_CLASS_NAME] {
        //
        // Methode principale
        //
        public [!output SAFE_CLASS_NAME]()
        {
        }
    }
}
```

Tous vos fichiers sont créés et bien présents. Vous pouvez alors ouvrir Visual Studio et créer une solution à partir de votre template de projet. Vous devriez alors avoir devant vous une solution fonctionnelle comprenant trois fichiers. Dans ceux-ci, le namespace est celui de votre projet, les noms des classes correspondant aux noms des fichiers sources et le fichier TODO.txt est personnalisé et comprend en titre, le nom du projet.

Conclusion

Vous venez de créer votre premier template de projet. Vous devriez rapidement apprendre à créer des templates évolués.

Il est possible de modifier les templates de base de Visual Studio mais vous risquez de tout casser. Agissez donc avec réflexion!

La dernière remarque consistera à vous rappeler que Visual Studio est très sensible à la casse des fichiers et/ou chemins d'accès. Faites donc attention à ce détail durant vos développements de templates.

Voici les fichiers utilisés lors de cet article: [Fichiers templates](#)

Remerciements

J'aimerais remercier tout particulièrement [Freereg](#) pour ses corrections apportées à cet article.