

# 1 Les assemblages

## 1.1 Présentation du problème

L'utilisation des bibliothèques partagées Microsoft est la cause de nombreux problèmes. Par exemple, la gestion des versions entre deux bibliothèques de même nom est impossible. Lorsqu'un logiciel installe la version 2 d'un DLL, il écrase la version 1 et impose aux autres logiciels l'utilisation de la nouvelle bibliothèque. Si celle-ci n'est pas pleinement compatible avec la version précédente, les logiciels moins récents ne fonctionneront probablement plus.

Pour pouvoir être accessible, les programmes et les composants doivent se décrire et s'enregistrer dans la base de registres. Les informations stockées deviennent rapidement obsolètes et incohérentes, et la base de registre finit par devenir une véritable usine à gaz souvent ingérable.

Enfin, pour qu'un service Web puisse accéder aux nouveaux services, il doit être arrêté puis redémarré afin que les classes COM utilisées par l'application soient mises à jour.

## 1.2 Solution

Pour résoudre ces problèmes, Microsoft a défini les 5 fonctionnalités de base suivantes pour les bibliothèques .NET :

- **Les applications doivent être auto descriptives.** Les applications qui sont auto descriptives suppriment le lien de dépendance avec le registre, ce qui permet une installation sans impact sur le système et simplifie la désinstallation et la réplication.
- **Les informations de version doivent être enregistrées et mises en oeuvre.** La prise en charge des versions doit être intégrée dans la plate-forme pour garantir que la version appropriée d'un lien de dépendance est chargée au moment de l'exécution.
- **Mémorisation des composants.** Lorsqu'une application s'exécute correctement, la plate-forme doit permettre la mémorisation du jeu de composants (y compris leur version) qui ont fonctionné ensemble.
- **Prise en charge des composants côte à côte.** La possibilité d'installer plusieurs versions d'un composant sur la machine et de les exécuter simultanément permet aux utilisateurs de spécifier la version qu'ils souhaitent charger, au lieu de se voir imposer une version. .NET Framework va plus loin dans cette gestion côte à côte des composants car il permet à plusieurs versions de la structure de travail elle-même de coexister sur une même machine. De ce fait, la mise à niveau est grandement simplifiée car un administrateur peut, si nécessaire, choisir d'exécuter des applications différentes sur des versions différentes de .NET Framework.
- **Isolement des applications.** .NET Framework doit faciliter l'écriture d'applications qui ne seront pas affectées par des modifications effectuées sur la machine par d'autres applications.

Les librairies .NET, appelées assemblages, possèdent ces fonctionnalités grâce à leur manifeste. Spécifiquement, un manifeste inclut les données suivantes sur l'assemblage :

- **Identité.** L'identité d'un assemblage comprend trois parties : un nom, un numéro de version et une culture (facultative).
- **Liste de fichiers.** Un manifeste inclut la liste de tous les fichiers qui composent l'assemblage. Pour chaque fichier, le manifeste enregistre son nom et un hachage cryptographique de son contenu au moment où il est créé. Ce hachage est vérifié au moment de l'exécution pour garantir que l'unité de déploiement est cohérente.
- **Assemblages référencés.** Les liens de dépendance entre les assemblages sont stockés dans le manifeste de l'assemblage appelant. Les informations sur les liens de dépendance incluent un numéro de version, utilisé au moment de l'exécution pour garantir que la version adéquate du lien de dépendance est chargée.
- **Ressources et types exportés.** Les options de visibilité disponibles pour les types et les ressources sont notamment "visibles uniquement dans mon assemblage" et "visibles pour les appelants externes à mon assemblage".
- **Demandes d'autorisation.** Les demandes d'autorisation pour un assemblage sont regroupées en trois catégories : 1) celles qui sont nécessaires pour que l'assemblage s'exécute, 2) celles qui sont souhaitables mais en l'absence desquelles l'assemblage aura tout de même accès à certaines fonctions, même si ces autorisations ne lui sont pas accordées, et 3) celles dont l'auteur ne veut absolument pas qu'elles soient accordées à l'assemblage.

En général, les assemblages sont composés de quatre éléments : les méta données de l'assemblage (manifeste), les méta données décrivant les types, le code en langage intermédiaire (IL) qui met en œuvre les types, et un ensemble de ressources. Tous ne sont pas présents dans chaque assemblage. Seul le manifeste est absolument obligatoire, mais ni les types ni les ressources ne sont indispensables pour attribuer à l'assemblage une quelconque fonctionnalité.

### ***1.3 Création d'une bibliothèque de classes.***

Pour créer une bibliothèque de classes, nous devons prendre l'option Nouveau – projet dans le menu Fichiers et nous choisissons alors bibliothèque de classes. Une classe est créée par défaut dans laquelle nous pouvons intégrer nos méthodes mais nous pouvons également ajouter de nouvelles classes.

Nous ajouterons une méthode dont le code sera le suivant:

```
public string MyFunction()
{
    return "Ma fonction version 1.0";
}
```

Nous pouvons changer le nom de l'assemblage lié à notre projet en ne perdant pas de vue que le nom de l'assemblage correspond au nom de notre fichier de sortie. Pour y arriver, il suffit d'éditer les propriétés de notre projet dans l'explorateur de solution. Nous retrouvons alors dans les propriétés communes générales la possibilité de changer le nom de notre assemblage sans que le nom du projet ne change nécessairement; nous choisirons comme nom de projet Myassembly et de ce fait, notre fichier de sortie prendra comme nom Myassembly.dll. Pour se clarifier les idées, nous retrouvons donc dans notre assemblage Myassembly un espace de nom qui est ClassLibrary1 et dans cet espace de nom nous avons une classe appelée Class1 comprenant la méthode 'MyFunction'.

Il suffit maintenant de générer la solution en appuyant sur les touches Ctrl Shift B, la bibliothèque de classes Myassembly.dll se trouvant alors dans le répertoire .../bin/release/.

Si nous voulons utiliser cet assemblage comme assemblage privé, nous n'avons plus d'intervention à effectuer sur notre code. Nous allons maintenant créer un nouveau projet dans lequel nous allons ajouter une référence à cet assemblage.

## **1.4 Référence à notre bibliothèque de classes.**

Nous allons recréer un nouveau projet Windows dans lequel nous allons ajouter une référence à notre bibliothèque de classes précédemment créée. Pour y parvenir, il suffit, après avoir créé notre projet de choisir l'option Ajouter une référence à partir de l'option du menu Projet.

Un assistant nous permet alors de choisir une nouvelle référence; nous cliquons sur le bouton parcourir pour choisir notre assemblage précédemment créé.

Une fois cette opération effectuée, nous pouvons retrouver une référence à cet assemblage ajouté à notre projet, en utilisant l'explorateur de solution. Si nous faisons apparaître le menu contextuel associé à notre assemblage et que nous prenons l'option propriétés, nous pouvons retrouver la propriété 'copie locale' à true indiquant qu'il s'agit d'un assemblage privé dont une copie doit se trouver dans le répertoire local de notre application.

Nous ajoutons dans notre formulaire un bouton et une zone de texte. Lorsque nous appuyons sur le bouton, un appel à la méthode 'MyFunction' doit être effectuée après avoir instancié la classe 'Class1' et la valeur renvoyée doit être affichée dans la zone de texte.

```
using ClassLibrary2;

private void button1_Click(object sender, System.EventArgs e)
{
    Class1 test = new Class1();
    this.textBox1.Text=test.MyFunction();
}
```

Pour bien se convaincre qu'il s'agit d'un assemblage privé, il suffit maintenant de supprimer le fichier Myassembly.dll de notre projet. La tentative d'exécution de notre code provoquera une exception.

## **1.5 Création d'un assemblage partagé.**

Nous reprendrons notre bibliothèque de classe en vue d'en créer un assemblage privé. Pour y parvenir, nous devons créer notre paire de clés en exécutant l'utilitaire sn.exe

La syntaxe sera dans notre exemple la suivante:

```
sn -k MyKey.snk
```

Le fichier 'MyKey.snk' contenant le jeu de clés sera alors créé, il suffira alors de le placer dans notre projet lié à notre bibliothèque de classes dans le répertoire /obj/release ou /obj/debug. Nous devons également éditer le fichier AssemblyInfo.cs pour y modifier le code suivant:

Ligne de code original: [assembly: AssemblyKeyFile("")]

Ligne de code modifié: [assembly: AssemblyKeyFile("MyKey.snk")]

Une fois cette opération effectuée, nous pouvons régénérer notre solution en appuyant sur les touches Ctrl Shift B. Il reste alors à enregistrer cet assemblage dans la cache globale: cette

opération peut s'effectuer grâce à la commande gacutil sous la syntaxe gacutil /i Myassembly.dll ou en utilisant dans les outils d'administrations le composant logiciel enfichable mscorecfg.msc.

Nous pouvons recréer un nouveau projet en ajoutant une référence non plus à un assemblage privé mais à un assemblage partagé. Alors que nous penserions retrouver notre assemblage enregistré dans la liste proposée par l'option d'ajout d'une référence, il n'en est rien du fait que cette liste se base sur des entrées présentes dans la base de registres et non pas dans la cache globale. Un article traitant de ce problème a été proposé par Microsoft sous le numéro Q306149. Malgré le fait que notre assemblage soit partagé, si nous devons en inclure une référence dans notre nouveau projet Windows, il faudra prendre l'option parcourir et choisir le fichier physique là où il se trouve dans notre disque dur. Une différence essentielle avec l'utilisation d'assemblages privés est que la propriété 'copie locale' de la référence ajoutée à notre projet est à false.

A priori, notre code fonctionne de la même façon. Si nous renommons ou supprimons le fichier Myassembly.dll, notre code continue à être exécutable, la preuve que le lien à cet assemblage s'effectue vers la cache globale des assemblages et non pas vers un le répertoire d'origine.

## 1.6 La gestion des versions.

Nous reprenons le projet associé à notre assemblage partagé et nous en modifions la version. Pour y parvenir, il suffit d'éditer le fichier AssemblyInfo.cs et d'y apporter les modifications suivantes:

Ligne de code original: `[assembly: AssemblyVersion("1.0.*")]`

Ligne de code modifié: `[assembly: AssemblyVersion("2.0.*")]`

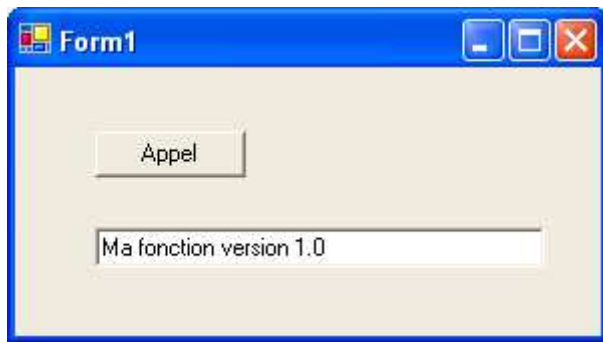
Nous pouvons alors régénérer notre solution et obtenir notre nouvel assemblage portant le même nom que notre assemblage précédent mais avec un numéro de version différent. Après l'avoir ajouté dans notre cache globale, nous retrouvons ces deux assemblages:



L'un possède bien la version 1.0.\* et l'autre 2.0.\*. Maintenant, qu'en est il de notre application Windows qui utilisais l'assemblage de la version 1.0? Pour le savoir, nous allons modifier le code de la version 2.0 de la façon suivante:

```
public string MyFunction()
{
    return "Ma fonction v2.0 appelée";
}
```

L'exécution de notre code provoque l'affichage suivant:



Notre application travaille toujours avec la version 1.0.\*