ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

[Tutoriel : Interfaçage Visual Studio (C#) -Excel]

Nicolas Chari, Thomas Dordonne, Aloys Fortier, Jonathan Zilmia

TABLE DES MATIERES

Introduction 4

Prérequis 5

Préparation de l'interfaçage 5

1. Téléchargement de Excel-DNA 5
2. 32 ou 64 bits ? 5

Partie I - Liaison Visual Studio - Excel 7

Configuration du projet 7

Fichier DNA 8

Import du fichier ExcelDna.xll 8

Mise en place du code 8

Partie II - Mise en place du debug 10

Configuration avancée permettant un debug en direct 10

Exécution en mode « Debug » 10

Partie III - Générer la release 12

Publication de la solution et export de la librairie 12

C#, ExcelDNA et VBA 13

1. Retourner dans Excel plus qu'une valeur 13
2. La même chose en VBA ? 13
3. Type de paramètres 13

Introduction

Pour diverses raisons, il peut être pratique, voir nécessaire de pouvoir relier un projet créé avec Visual Studio (en C# dans notre cas, mais ce tutoriel est compatible avec d'autre langages comme le C++ ou le F#) à Excel. Dans notre cas, des applications financières développées en C# doivent impérativement être reliées à Excel, pour l'import des données et l'exportation des résultats. La partie « C# » servant principalement au traitement des données disponibles dans des Spreadsheets Excel, et à effectuer des calculs difficilement adaptables en VBA.

Dans ce tutoriel, nous utilisons la librairie Excel-DNA plutôt que la librairie Interop, certes plus connue, mais qui nécessite d'avoir Visual Studio installé sur l'ordinateur utilisant notre programme. Un des avantages non négligeable d'Excel-DNA, est que la release est exécutable sur un ordinateur ne disposant que d'Excel.

L'interfaçage va créer une librairie Excel (format XLL). Grossièrement, le projet d'interfaçage va fournir un « fichier » contenant des fonctions Excel (correspondant au code C#), et pouvant être utilisé par Excel de façon totalement indépendante.

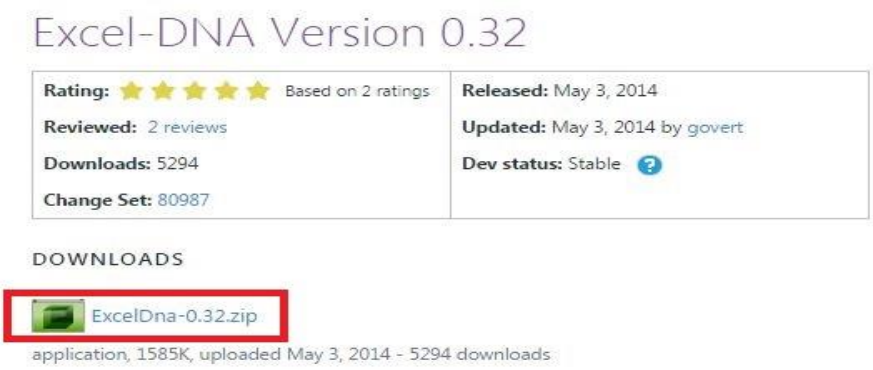
La librairie ExcelDna fonctionne sur processeur 32 et 64 bits, Excel 1997-2013 et Visual Studio 2008-2013.

Prérequis

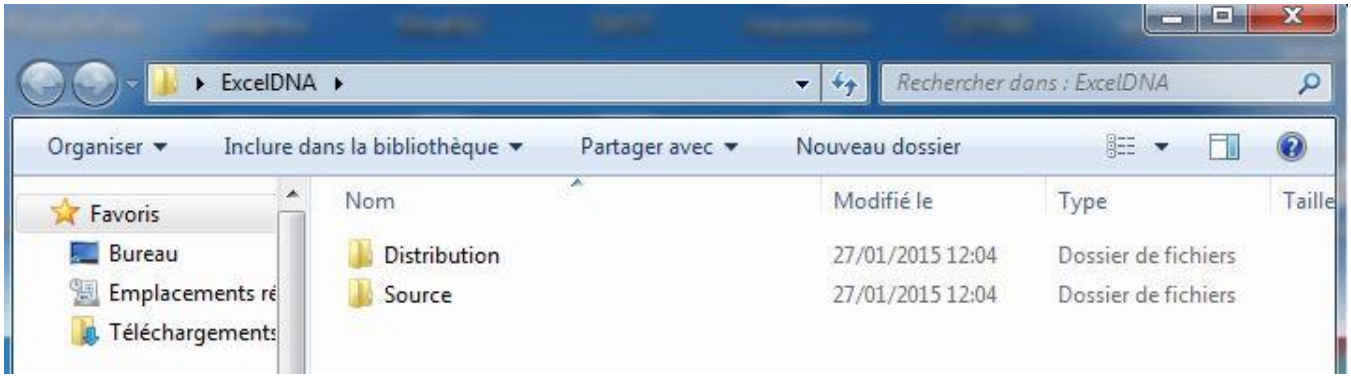
Préparation de l'interfaçage

1. Téléchargement de Excel-DNA

La première étape consiste à télécharger la dernière version d'Excel-DNA en suivant le lien suivant http://exceldna.codeplex.com/ dans l'onglet Downloads.

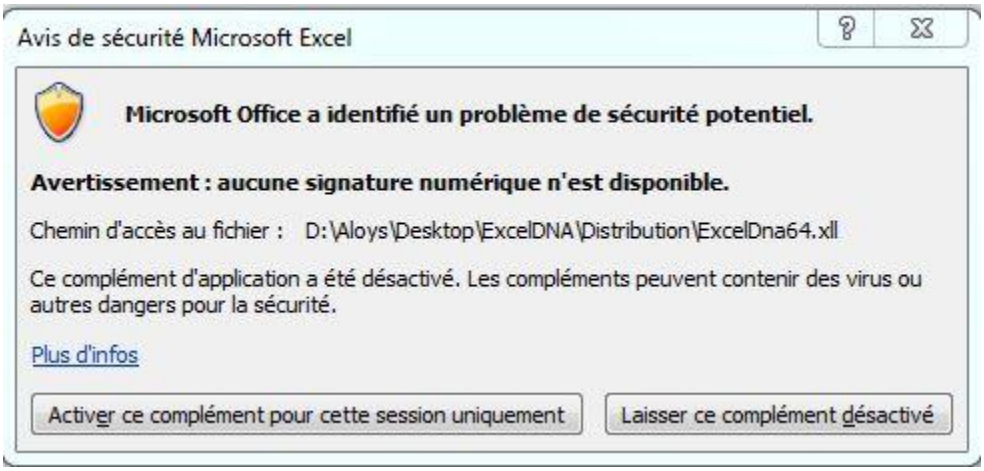


Créez un nouveau dossier ExcelDNA. Ensuite il faut extraire les dossiers distribution et source de l'archive dans le dossier nouvellement créé. Vous devriez obtenir un dossier ExcelDna composé comme ceci :

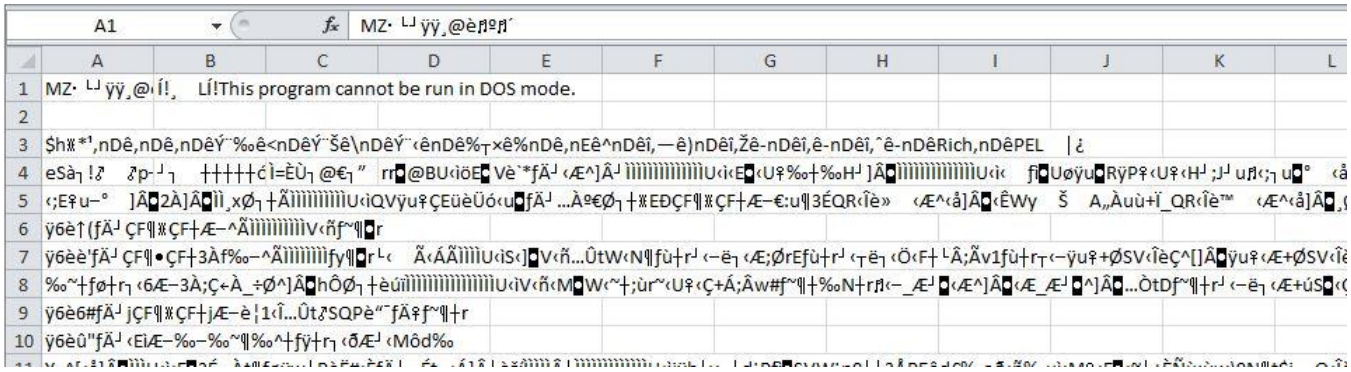


2. 32 ou 64 bits ?

La deuxième étape consiste à tester 2 fichiers pour savoir quelle version d'Excel-DNA utiliser selon votre version d'Excel. Pour cela, dans le dossier ExcelDNA, dans Distribution, vous devriez trouver deux fichiers au format XLL, nommés ExcelDna et ExcelDna64. Double-cliquez sur ExcelDna. Excel s'ouvre et vous demande si vous voulez activer ce complément, répondez oui.



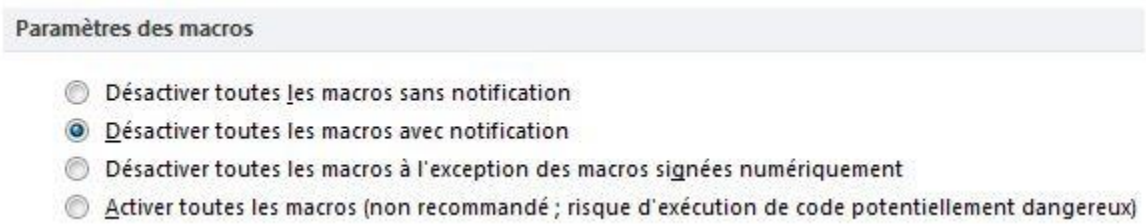
Si rien ne se passe, la version à utiliser est 32 bits. En revanche, si vous obtenez un rendu tel que sur le screenshot ci-dessous, vous devez choisir la version 64 bits.



Si la version 32 bits semble convenir, essayez quand même de lancer le fichier Exceldna64 pour « vérifier » qu'il ne s'ouvre pas correctement, et inversement, si la version 32 bits pose problème, exécutez le fichier Exceldna64 pour confirmer que le complément fonctionne.

A cette étape du tutoriel, vous devez donc savoir quel fichier choisir entre Exceldna.xll et Exceldna64.xll

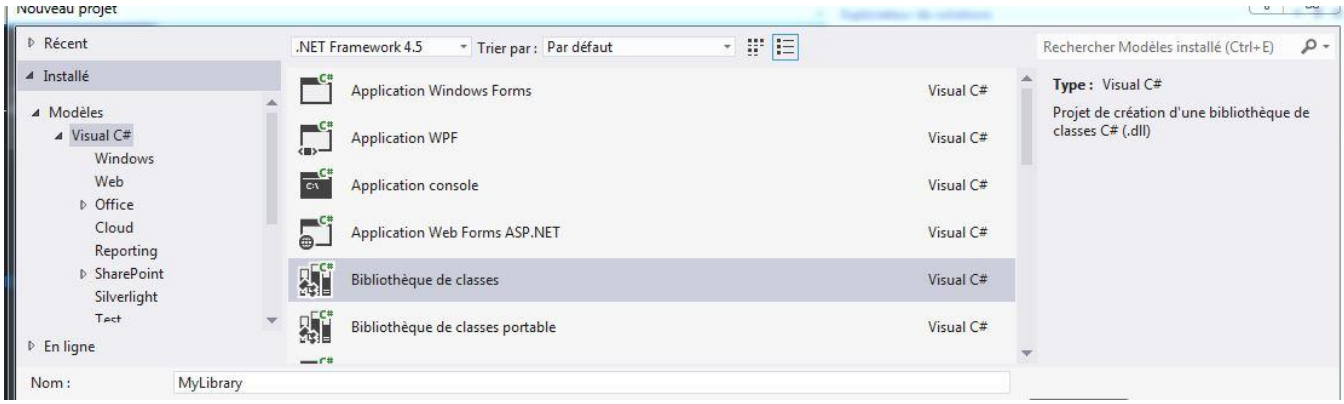
Ps : Par défaut, Excel vous demande si vous voulez exécuter un complément dont la source est inconnue. Si toutefois Excel ne semble pas ouvrir le fichier, et n'affiche aucun popup de confirmation, vérifiez dans les paramètres d'Excel (paramètres Macro) que vous n'avez pas la première option de cochée, la deuxième option étant nécessaire.



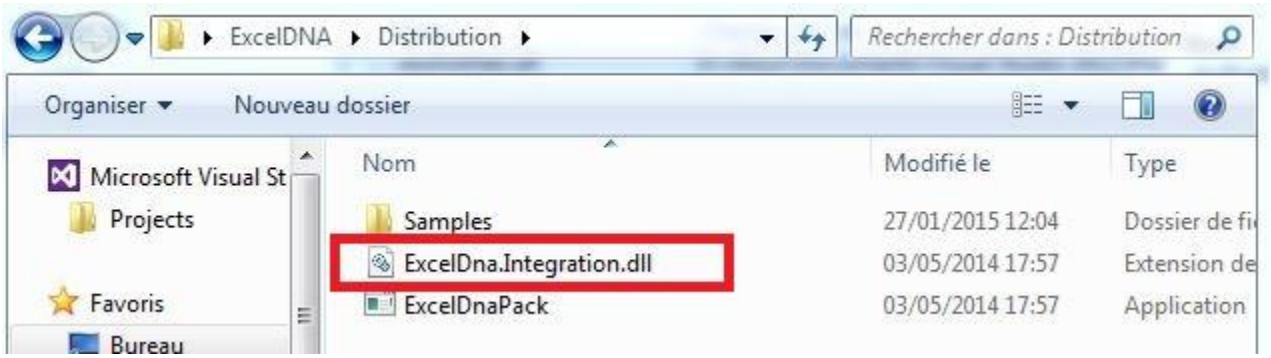
Partie I – Liaison Visual Studio - Excel

Configuration du projet

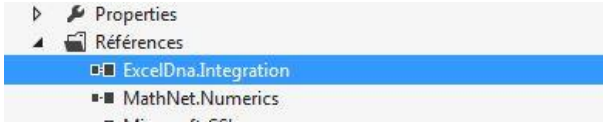
Dans Visual Studio, créez un nouveau projet de type Bibliothèque de classe. Appelez le MyLibrary.

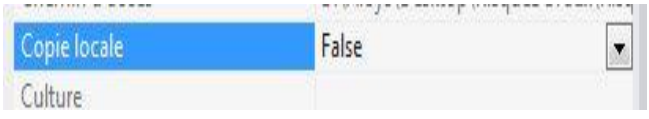


Ajoutez ensuite une référence vers ExcelDna. Pour se faire, dans Visual Studio, cliquez sur Projet -> Ajouter une référence -> Parcourir. Puis récupérez dans le dossier ExcelDna/Distribution le fichier ExcelDna.Integration.dll



Dans votre projet Visual Studio, dans l'explorateur de solution, vous pouvez cliquer sur Références -> ExcelDna.Integration pour afficher les paramètres affectés à cette référence. Dans Copie locale, sélectionnez False





Fichier DNA

Avec un éditeur de texte (Notepad++ par exemple), créez un nouveau fichier sans extension. Recopiez les 3 lignes ci-dessous à l'intérieur, et sauvegardez le en tant que : « FirstAddIn.dna »(l'extension doit être .dna et pas .txt)

```
<DnaLibrary Name="First Add-In" RuntimeVersion="v4.0">
  <ExternalLibrary Path="MyLibrary.dll" />
</DnaLibrary>
```

Ajoutez ensuite ce fichier à votre projet. (Dans Visual Studio, Clic-droit sur le nom du projet -> Ajouter -> Elément existant). De la même manière que pour la référence, cliquez une fois sur le fichier pour afficher ses propriétés, et pour l'option « Copier dans le répertoire de sortie », sélectionnez « Copier si plus récent ».

Import du fichier ExcelDna.xll

Dans les premières étapes de ce tutoriel, vous aviez choisi la version (32 ou 64 bits) du fichier ExcelDna.xll adapatée à votre configuration. Séléctionnez le bon fichier, copiez-le et renommez-le « FirstAddIn.xll ». Ajoutez ensuite ce fichier à votre projet (Dans Visual Studio, Clic-droit sur le nom du projet -> Ajouter -> Elément existant). De la même manière que pour le fichier FirstAddIn.dna, dans ses propriétés, choisissez de le copier dans le répertoire de sortie si plus récent.

Mise en place du code

Dans votre projet Visual Studio, ouvrez le fichier Class1.cs. C'est à partir de cette classe que se fera la communication avec Excel en mode Debug. La classe « Class1.cs » servira en quelque sorte de « main » autrement dit le fichier habituellement appelé « Program.cs ».

Au début de votre classe, ajoutez un using pour la référence ExcelDna en tapant « using ExcelDna.Integration ; ». Placez ce using à la suite des using déjà existants.

```
using System.Text;
using System.Threading.Tasks;
using ExcelDna.Integration;
```

Déclarez ensuite une fonction « test » comme ceci :

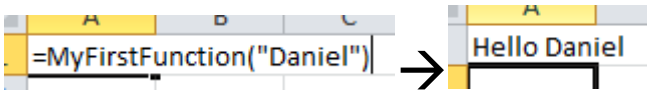
```
namespace MyLibrary
{
    public class Class1
    {
        [ExcelFunction(Description="Ma première function ExcelDna")]
        public static string MyFirstFunction(string name)
        {
            return "Bonjour " + name;
        }
    }
}
```

La description est le texte qui sera affiché dans Excel, si vous affichez la description de votre fonction.
Le nom de la fonction « MyFirstFunction » est celui qui devra être tapé dans la barre de formule Excel pour exécuter le code C#. Cette fonction prend en paramètres un string.
Lors de l'éxécution du code, la fonction affichera Bonjour + le texte dans la cellule Excel d'où on appellera la fonction.

Si vous ne souhaitez pas mettre en place le Debug, vous pouvez d'ores et déjà compiler le projet et générer la solution.
Dans le dossier de votre projet -> bin -> Debug, exécutez le fichier FirstAddIn.xll. Excel s'ouvre, vous pouvez tapez la formule suivante dans une cellule

=MyFirstFunction(« ESILV »)

Le résultat devrait être « Bonjour ESILV ».

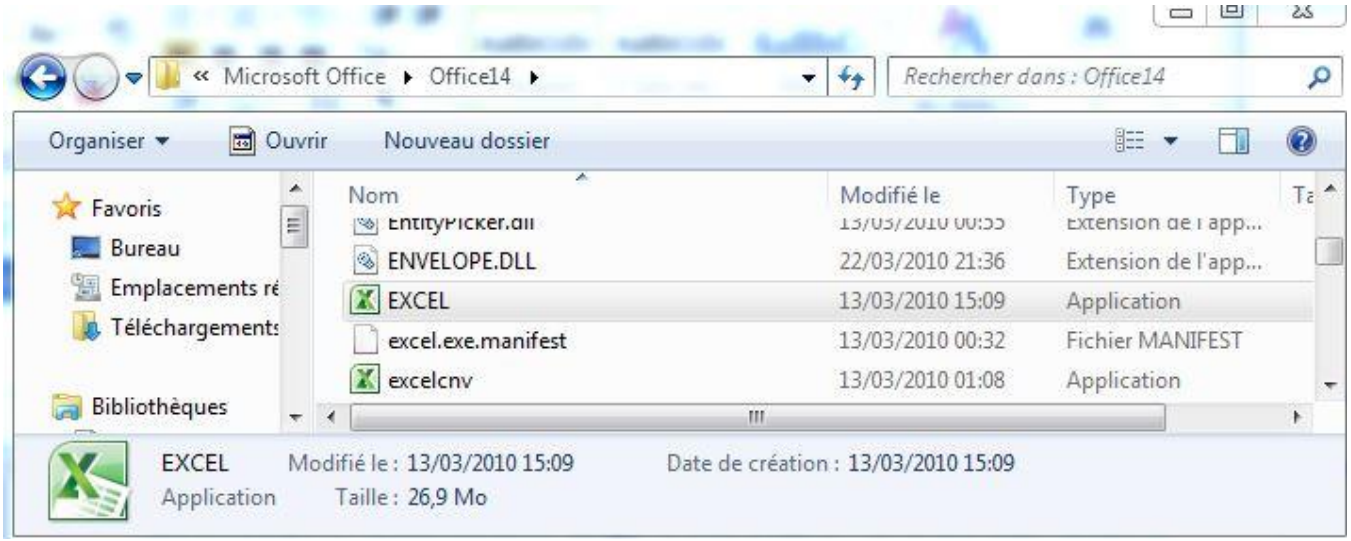


Partie II – Mise en place du debug

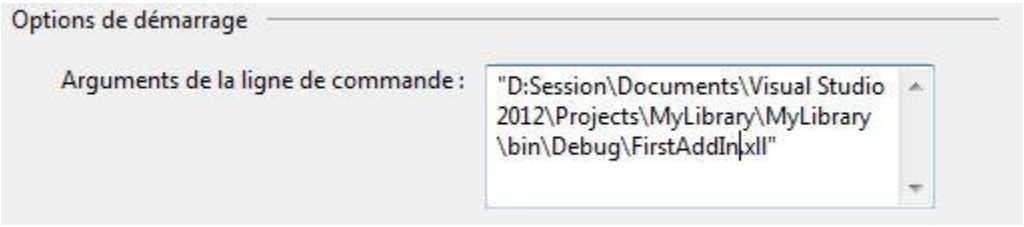
Configuration avancée permettant un debug en direct

Il peut être pratique voir nécessaire de pouvoir débugger en direct le code C#. Pour se faire, rendez vous dans les propriétés du projet, dans l'onglet Déboguer.

Dans la partie « Action de démarrage », Cochez « Démarrer le programme externe ». Cliquez ensuite sur parcourir pour indiquer le chemin vers Excel. Par défaut, il se situe dans ProgramFiles -> Office14 (pour Excel 2010) ou Office12 (Pour Excel2007).



Ensuite, dans Options de démarrage, dans la partie Arguments de la ligne de commande, renseignez le chemin vers le fichier généré lors de la compilation du projet. Le fichier s'appelle FirstAddIn.xll et se situe dans le dossier de votre projet /bin/debug



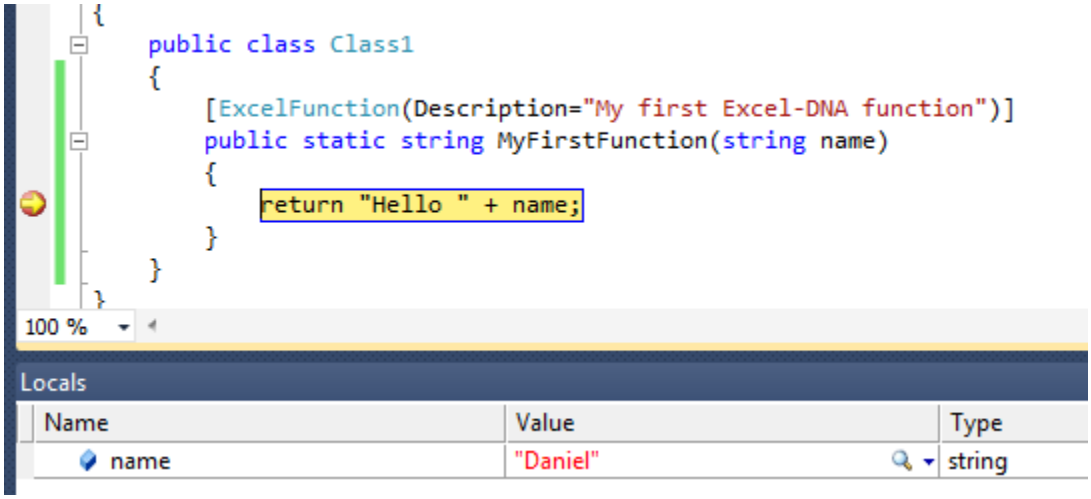
Exécution en mode « Debug »

Une fois la configuration réalisée, vous pouvez simplement mettre un point d'arrêt dans la fonction « MyFirstFunction »

```
public static string MyFirstFunction(string name)
{
    return "Hello " + name;
}
}
```

Générez et lancez le projet en appuyant sur F5. Excel s'ouvre et charge la librairie. Créez ou ouvrez un classeur Excel. Exécutez la fonction comme nous l'avons vu dans la partie précédente

Avant d'afficher le résultat, le point d'arrêt devrait être atteint.



Le debug vous permet donc de vérifier votre code et vos résultats en direct.

Partie III – Générer la release

Publication de la solution et export de la librairie

Une fois votre projet terminé, la solution peut être exporté sous la forme d'un seul fichier au format xll. Une « bibliothèque » de fonctions, développées en C# mais exécutables sur Excel. Pour se faire, dans le fichier FirstAddIn.dna de votre projet, modifez le en ajoutant un directive de packing pour le fichier MyLibrary.dll. Votre fichier devrait être comme ceci :

```
<DnaLibrary Name="First Add-In" RuntimeVersion="v4.0">
  <ExternalLibrary Path="MyLibrary.dll" Pack="true"/>
</DnaLibrary>
```

Nettoyez la solution et Compilez le projet en mode « release »

Ouvrez l'invite de commande Windows, dans le dossier de la release (MyLibrary/bin/release).

Il suffit d'exécuter ExcelDnaPack.exe qui est dans le dossier ExcelDna/distribution, pour qu'il « pack » la release.

En exécutant la ligne de commande suivante : « C:\Session\Desktop\ExcelDna\Distribution\ExcelDnaPack.exe FirstAddIn.dna »

```
C:\Test\MyLibrary\bin\Release>C:\Test\ExcelDna\Distribution\ExcelDnaPack.exe FirstAddIn.dna
Using base add-in FirstAddIn.xll
  ->  Updating resource: Type: ASSEMBLY_LZMA, Name: EXCELDNA.INTEGRATION, Length: 43546
  ~~> ExternalLibrary path MyLibrary.dll resolved to C:\Test\MyLibrary\bin\Release\MyLibrary.dll.
  ->  Updating resource: Type: ASSEMBLY_LZMA, Name: MYLIBRARY, Length: 1514
  ->  Updating resource: Type: DNA, Name: __MAIN__, Length: 385
Completed Packing FirstAddIn-packed.xll.

C:\Test\MyLibrary\bin\Release>
```

Le résultat final est un fichier « FirstAddIn-packed.xll » qui peut être exécuté sans Visual Studio, renommé, distribué selon vos envies.

C#, ExcelDNA et VBA

Maintenant que votre projet est lié à Excel et que tout fonctionne correctement, il peut être utile d'avoir quelques astuces en VBA.

1. Retourner dans Excel plus qu'une valeur

Si votre fonction C# retourne un tableau de valeurs, il faudra déclarer le type de retour de la fonction C# comme `Object[,]`

```
[ExcelFunction(Description = "My first E
public static Object[,] MyFirstFunction(
{
```

Coté Excel, cela revient à exécuter une formule matricielle. Si votre tableau C# à une dimension de 5 lignes et 3 colonnes par exemple, sélectionnez une plage de cette taille, tapez la formule (« =MyFirstFunction(parametreA,…ParametreN) ») puis Ctrl+Shift+Entrée. Votre range est ainsi rempli des valeurs contenues dans le tableau retourné par le C#.

2. La même chose en VBA ?

Pour exécuter le code C# sans avoir à retaper le nom de la fonction à chaque fois, ou tout simplement pour automatiser les tâches, un peu de VBA est nécessaire.

Pour un résultat unique, le code VBA est le suivant :

Sheets(« NomFeuille »).Cells(i,j) .Formula = « = MyFirstFunction(" & VariableVBA & ")"

Pour un résultat d'un tableau comme décrit précédemment, le code VBA est le suivant :

Range("B9:D10").FormulaArray = " =MyFirstFunction(" & VariableVBA1 & "," & VariableVBA2 & ")"

3. Type de paramètres

Attention au symbole décimal. Par défaut, il faut que ce soit une virgule sous Excel, qui sera converti en point en C#. Vous pouvez modifier le symbole décimal dans les paramètres Excel pour que ce soit un point.

Vous pouvez envoyer les paramètres des types gérés par le C#.

Si vous passez en paramètre de votre fonction un Range, le type correspondant en C# est Object[,].

Les types Integer, Double, et String en VBA, correspondent respectivement aux types int, double et string en C#

Enfin, n'oubliez pas que le fichier Class1.cs est le fichier de « communication » avec Excel. Vous pouvez ajouter autant de classes que vous le souhaitez à votre projet.