Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Introduction à WPF

Microsoft Student Partners

Julien DOLLON

Sommaire

1 Introduction

Pour la plupart des utilisateurs, une application est juste et tout simplement une fenêtre qui s'ouvre et qui leur permet d'interagir avec l'ordinateur. Mais pour le développeur, la gestion de cette fenêtre n'est pas forcément des plus simples. En effet il faut choisir une librairie adaptée, …

Pour ce qui est de .Net, on a le choix entre les WinForms, et depuis la sortie du Framework .Net 3.0 (2006) avec Windows Vista : WPF ; Même si comme on pourra le voir, ce dernier complète davantage les composants WinForms. On va pouvoir voir dans ce premier chapitre d'introduction à WPF, en quoi consiste WPF, ce qu'il nous apporte et comment l'utiliser.

1.1 Avant WPF

Comme on a pu l'énoncer précédemment, WPF est très récent étant donné qu'il est apparu avec le Framework .Net 3.0. Avant nous n'avions que les WinForms.

Les WinForms, c'est le nom donné à la partie du Framework .Net, responsable de la partie interface utilisateur (GUI). Pour ceux d'entre vous qui avez déjà développé avec le langage Visual Basic 6, les WinForms apparaissent similaires aux Forms de ce même langage, tout en ayant apporté leur lot d'avantages. En effet, elles sont très faciles à prendre en main et très orienté objet.

1.2 Pourquoi WPF

A la sortie de Windows Vista, on voit apparaitre des effets 3D et en même temps la sortie du .Net Framework 3.0. Pourquoi ? On s'est aperçu que les WinForms n'étaient pas vraiment adaptés (pour de nombreuses raisons techniques telles qu'elles ne sont pas forcément des plus aisées à personnaliser). Elles posent également un problème au niveau du travail collaboratif entre designers et développeurs, et bien d'autres sur lesquelles nous aurons l'occasion de revenir dans de futurs chapitres.

On verra tout au long de nos chapitres, que WPF apporte son lot de nouveautés qui facilitent le « design » de la GUI. Par exemple, on peut citer les graphismes vectoriels, la transparence par pixel, les animations, l'adaptation à la résolution, le support des templates de data binding (on aura l'occasion d'y revenir dans les chapitres futurs), et bien d'autres…

2 Les plus et les moins

Comme n'importe quelle technologie, WPF a des avantages comme des inconvénients, même ces derniers sont malgré tout très restreints, comme on va pouvoir le voir dans les chapitres qui suivront.

2.1 Les avantages de WPF

2.1.1 Utilisation du GPU

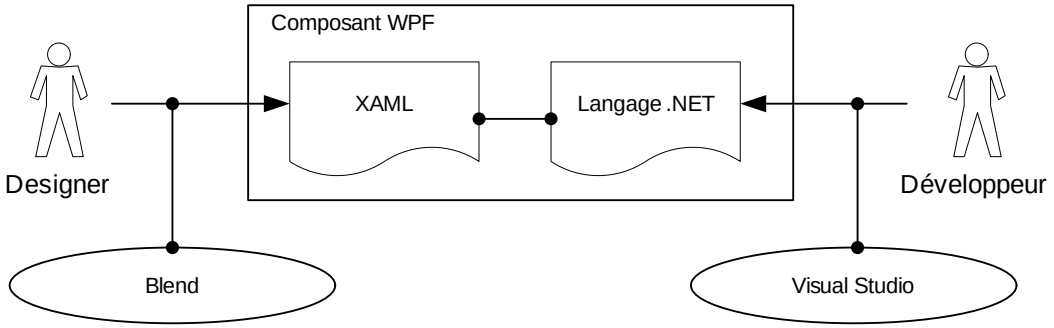
Un des principaux changements avec WPF c'est l'utilisation du GPU. Bref rappel de base, le GPU est en fait le processeur graphique présent sur votre carte graphique.

Le fait que WPF utilise le GPU change énormément de choses. En effet cela permet de déléguer une partie du travail habituellement délégué au microprocesseur (CPU), au processeur graphique qui va se charger de la manipulation de données graphiques. Ce n'est pas votre GPU qui va se charger des traitements conditionnels... Non pas qu'il ne puisse pas le faire, mais tout simplement parce qu'il n'est pas optimisé pour de tels traitements.

Les GPU étant de plus en plus puissant et nos applications de plus en plus « jolies » et par conséquent lourdes… Vous comprendrez que le fait que WPF utilise le GPU n'est pas inutile tout au contraire.

2.1.2 Séparation code / design

Vous avez souvent dû vous en rendre compte, lorsque vous développez une application en collaboration avec des designers et d'autres développeurs, un problème majeur va se poser. En effet pour customiser votre application, votre designer devra avoir des compétences en développement, il va devoir connaître les objets de votre application et les fonctionnalités des WinForms… Vous l'aurez compris au final vous allez devoir adapter tout ça vous-même…



C'est là que WPF vous permet de justement séparer en couches votre application. Si vous avez une certaine expérience en développement, vous avez surement l'habitude d'ors et déjà de séparer votre application manuellement en plusieurs couches, comme par exemple avec le modèle logiciel MVC.

WPF va se charger de séparer le code designer du code behind (classe d'arrière plan). C'est-à-dire que le designer va pouvoir travailler sur le design de l'application, via un langage commun basé

sur du XML qui est le XAML qu'on va pouvoir voir par la suite. Quant au développeur de son côté via le code behind il va pouvoir travailler sur la couche métier. Cela va permettre une meilleure productivité et un support de l'application plus facile par la suite.

2.1.3 Plus puissant que les WinForms

Même si au premier abord, WPF peut choquer par la séparation code / design et aux contrôles également sensiblement différents, WPF offre plus de possibilités comme on a pu le voir et plus de « puissance ».

C'est-à-dire que par exemple grâce à WPF et Expression Blend on va pouvoir mettre en place des animations pour notre application, de façon vraiment très simple et très rapide. Cela augmente d'une part la productivité du développeur, celle du designer et pour finir l'ergonomie de l'application pour l'utilisateur final.

De plus on peut parler de la puissance de WPF au niveau du Data Binding (liaisons de données). En effet le Data Binding est un mécanisme puissant. Nous verrons plus en détail en quoi consiste le Data Binding plus en détail.

2.2 Les inconvénients

2.2.1 Manque d'interopérabilité

Le principal problème de WPF reste le même que les WinForms c'est-à-dire l'interopérabilité de ce dernier. En effet on ne peut pas, dans l'état actuel des choses, faire du WPF sous linux sous Mac OS ou d'autres systèmes… Le seul portage de .NET fait sur les autres systèmes reste mono (je vous invite à consulter les cours de mono de dotnet-france) et mono ne supporte actuellement pas le WPF.

On peut dire que cela est appelé à changer dans le temps, car quand on regarde du côté de Silverlight, qui est « l'équivalent » de WPF pour les applications web .NET, on peut remarquer qu'il y a un portage réalisé qui s'appelle Moonlight… Donc, à suivre.

2.2.2 Manque de maturité

Egalement on peut citer un inconvénient normal qui est le manque de maturité de WPF. En effet, WPF est arrivé avec le Framework .NET 3.0. C'est une technologie qui a actuellement plus de 2 ans (depuis 2006). Ainsi, les communautés concernant WPF restent peu développées, et les entreprises commencent à peine à se pencher dessus.

Heureusement, nous pensons que cet inconvénient sera amené à disparaître dans le temps.

2.2.3 Tout est à refaire

Enfin, on peut dire que la migration des applications WinForms actuelle reste complexe.

C'est pourquoi, le développeur qui souhaite migrer son application WinForms vers WPF, va devoir revoir et recréer toute sa GUI. Pour peu de vouloir utiliser les spécificités de WPF, il faudra également qu'il revoit son architecture, notamment pour l'utilisation du DataBinding.

3 Notions de bases

3.1 Les outils

Si vous avez développé des applications utilisant les Windows Form auparavant, vous avez du utiliser Visual Studio. WPF bouscule un peu cette habitude avec l'introduction du XAML. Dorénavant vous allez pouvoir gérer d'une part la partie graphique de votre application avec n'importe qu'elle application gérant le XAML, et d'autre part la partie code avec votre IDE préféré.

Pour écrire ce cours, nous utilisons deux logiciels conseillés par Microsoft : Visual Studio 2008 Sp1 pour écrire le code de gestion de l'application, et Expression Blend 2 Sp1 pour la partie graphique.

Visual Studio est un environnement de développement complet proposé par Microsoft. A l'heure ou nous écrivons ce chapitre, vous pouvez vous procurer Visual Studio 2008 dans ses versions Standard, Professionnelle et Team System. Si vous possédez une version antérieure à Visual Studio 2008, sachez que le support natif de WPF n'est pas implanté.

Pour pouvoir développer dans de bonnes conditions, nous vous conseillons d'installer le Sp1 du Framework 3.5 incluse dans le SP1 de Visual Studio, qui améliore grandement le designer d'application embarqué, et améliore de 20 à 45% les performances de vos applications WPF, sans modifier une ligne de code.

Si vous ne pouvez pas vous procurer Visual Studio 2008, vous pouvez vous rabattre sur Visual C# Express disponible gratuitement à cette adresse : http://www.microsoft.com/express/vcsharp/

Si Visual Studio 2008 gère parfaitement bien le développement d'applications avec les langages C#, le VB.NET et le XAML, il est cependant trop « simple » pour le coté graphique. Pour combler ce manque, Microsoft a développé Expression Blend, de la suite logicielle Expression.

Expression Blend, est à l'origine un outil permettant de créer des graphiques vectoriels. Il a été étendu à l'occasion de la sortie de WPF afin de gérer le XAML nativement. Grâce à Expression Blend, un graphiste peut travailler directement sur une solution Visual Studio sans toucher une seule ligne de code. Expression Blend gère également la création d'animations totalement en XAML. Pour ce cours, nous avons utilisé Expression Blend 2 Sp1 qui gère WPF et Silverlight 2.

Sachez enfin que vous pourrez utiliser n'importe quel logiciel du marché permettant d'exporter vos travaux en XAML, à la place d'Expression Blend. Par exemple, il existe dors et déjà des extensions à Maya vous permettant d'exporter des modèles 3D en XAML.

3.2 Système d'exploitations compatibles

Les applications WPF fonctionnent sur toutes les machines dotées du Framework .NET 3.0. Parmi les systèmes d'exploitation supportant ce Framework vous trouvez :

- Windows XP Service Pack 2 ou ultérieur
- Windows Server 2003 Service Pack 1 ou ultérieur
- Windows Vista
- Windows Server 2008

Note : A partir de Windows Vista, le Framework .NET 3.0 est installé par défaut.

Cela étant dit, nous vous conseillons d'installer le Framework .NET 3.5 Sp1, afin de bénéficier de l'amélioration de performances apportées.

4 L'intérêt du XAML

4.1 Pourquoi XAML ?

Le XAML (eXtensible Application Markup Language) est un langage déclaratif basé sur la syntaxe du XML. Il permet grâce à des balises et des attributs de créer très facilement des objets. Pour cela, le compilateur XAML se charge de déclarer et définir des objets dynamiquement grâce aux balises (équivalent des classes) et aux attributs (équivalents aux propriétés) XAML.

Malgré sa syntaxe simple, le XAML permet de restituer des graphiques vectoriels, ou des modèles 3D aisément. Les possibilités graphiques sont donc infinies.

Il existe quelques règles élémentaires, issues de la syntaxe du XML, qu'il vous faudra respecter si vous faites du XAML :

- Respecter la casse.
- Les balises ouvertes doivent être refermées sans se chevaucher.
- Chaque attribut doit obligatoirement avoir une valeur inscrite entre guillemets ou apostrophes.

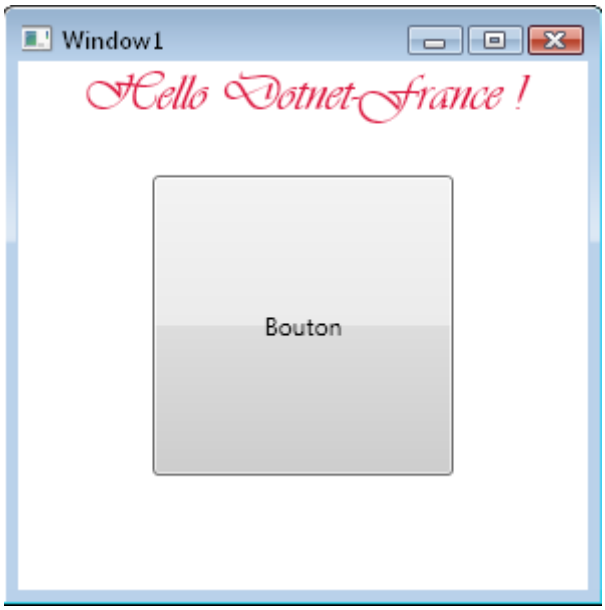
Note : Voir la définition d'une image vectorielle sur Wikipedia pour mieux comprendre : http://fr.wikipedia.org/wiki/Image_vectorielle

4.2 Exemple

Nous allons étudier ensemble un court exemple, afin de voir à quoi ressemble la syntaxe du XAML. Nous allons simplement créer un bouton accompagné d'un texte. Pour l'instant, l'exemple à pour but de vous familiariser avec le XAML, nous n'allons pas entrer dans une description détaillée du code.

```
<!—XAML-->
<Window x:Class="Wpf.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
        <Button Width="150" Height="150">Bouton</Button>
        <TextBlock Text="Hello Dotnet-France !"
                   FontSize="26"
                   FontFamily="Vivaldi"
                   HorizontalAlignment="Center"
                   Foreground="Crimson" />
    </Grid>
</Window>
```

Si nous compilons et exécutons ce code, voici le résultat :



Détaillons le code par morceaux :

```
<!–XAML-->
<Window x:Class="Wpf.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
</Window>
```

Ce bout de code permet d'instancier une nouvelle fenêtre. Celle-ci aura comme titre Window1, d'une hauteur de 300 pixels, et d'une largeur de 300 pixels. Les valeurs des attributs xmlns et xmlns :x référencent deux espaces de nom, qui contiennent la plupart des balises du XAML. Ces deux valeurs seront nécessaires pour instancier correctement nos différents objets.

Dans le bloc de code suivant, nous commençons à voir l'esquisse d'un bouton et un block de texte. Ces deux contrôles sont contenus dans un élément Grid. Même si nous reviendrons sur cette balise dans un prochain cours, sachez qu'elle sert ici de conteneur à divers contrôles.

```
<!–XAML-->
<Grid>
    <Button Width="150" Height="150">Bouton</Button>
    <TextBlock Text="Hello Dotnet-France !"
               FontSize="26"
               FontFamily="Vivaldi"
               HorizontalAlignment="Center"
               Foreground="Crimson" />
</Grid>
```

```
<!–XAML-->
<Button Width="150" Height="150">Bouton</Button>
```

Dans notre grid nous avons tout d'abord notre bouton, créé simplement en ajoutant la balise Button, auquel on passe divers attributs, ici la hauteur et la largeur. La valeur entre les deux balises Button correspond dans le cas présent à la valeur de l'attribut Content. Nous aurions donc pu écrire cet exemple :

```
<!—XAML-->
<Button Width="150" Height="150" Content="Bouton" />
```

Note : Si dans le cas du bouton la valeur entre les balises correspond à l'attribut Content, ce n'est pas le cas pour toutes les balises.

```
<!—XAML →
<TextBlock Text="Hello Dotnet-France !"
           FontSize="26"
           FontFamily="Vivaldi"
           HorizontalAlignment="Center"
           Foreground="Crimson" />
```

Enfin nous avons créé un contrôle TextBlock, qui va afficher « Hello Dotnet-France ! » avec la police Vivaldi, de taille 26 et de couleur Crimson.

Dans cet exemple, nous avons créé quatre objets différents, une fenêtre, une grille (grid), un bouton, et un textblock en XAML. Pour vous convaincre de la simplicité et du pratique de ce morceau de code, nous allons « convertir » intégralement notre code, en C#:

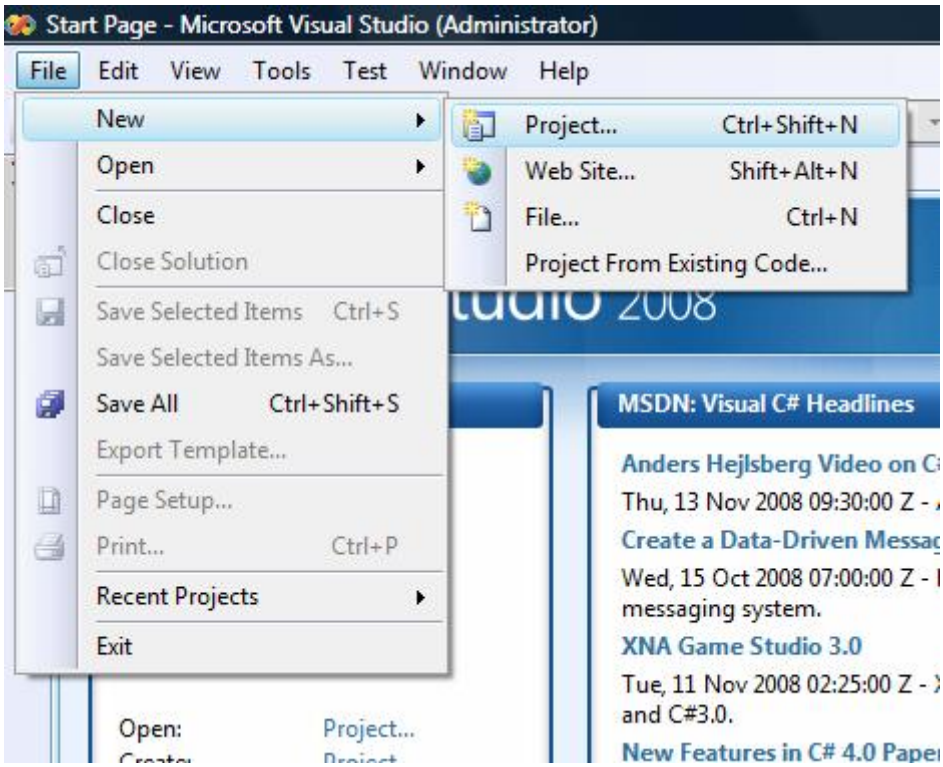
```
//C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Controls;
using System.Windows;
using System.Windows.Media;

namespace Wpf
{
    class Exemple
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Button b = new Button();
            b.Content = "Bouton";
            b.Width = 150;
            b.Height = 150;

            TextBlock text = new TextBlock();
            text.Text = "Hello Dotnet-France !";
            text.FontSize = 26;
            text.FontFamily = new FontFamily("Vivaldi");
            text.HorizontalAlignment = HorizontalAlignment.Center;
            text.Foreground = Brushes.Crimson;

            Grid grille = new Grid();
            grille.Children.Add(b);
            grille.Children.Add(text);

            Window window = new Window();
            window.Title = "Window1";
            window.Height = 300;
            window.Width = 300;
            window.Content = grille;

            Application app = new Application();
            app.Run(window);
        }
    }
}
```

L'exemple parle de lui-même, le XAML nous fait vraiment gagner en simplicité, et ce même sur un exemple simple. Cela étant, vous pouvez avoir besoin de créer certains objets en C#, notez donc que tout ce qui est faisable en XAML est faisable en C#, en VB.NET et dans tous les langages managés gérés par le Framework .NET en général.

5 Premier projet

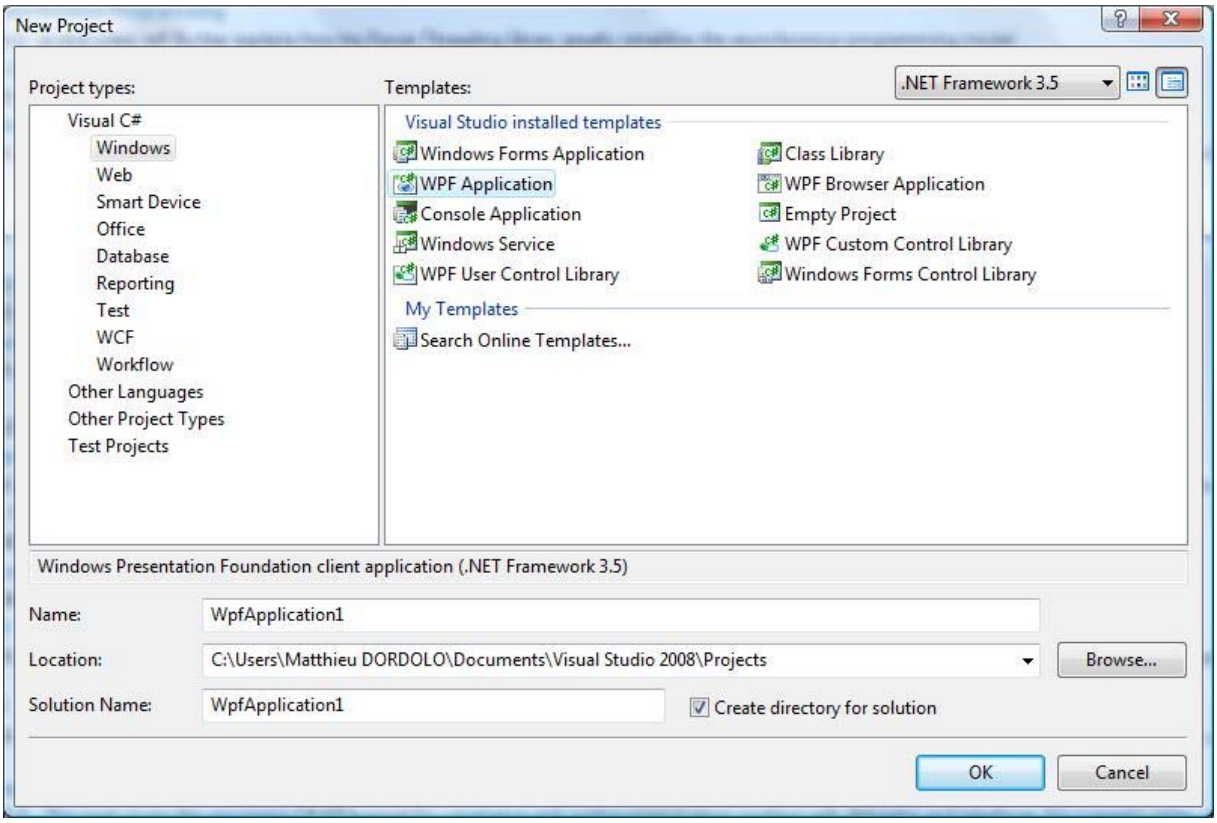
5.1 Création d'un projet

Pour ce qui est de la création de projet WPF cela ne diverge pas trop de la création de projet WinForms classique :



Dans le menu de Visual Studio : Fichier->Nouveau->Projet ou le raccourci clavier Ctrl+Shift+N

Ensuite vous avez le choix entre plusieurs types de projets WPF :



Comme vous pouvez le voir, les applications WPF et Windows Forms sont regroupés dans le même menu car le but final reste le même.

Nous avons 4 types de projet WPF disponibles qui sont :

- WPF Application
- WPF Browser Application
- WPF Custom Control Library
- WPF User Control Library

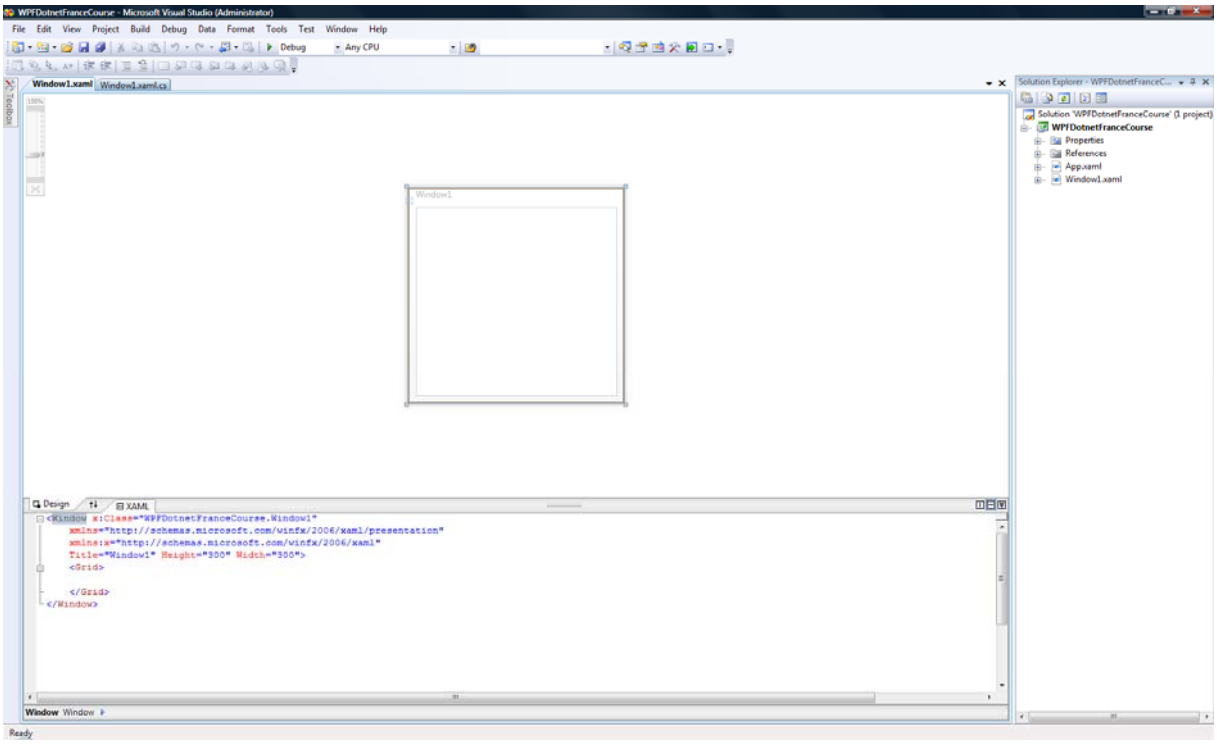
Une Application WPF est donc tout simplement une application WPF basique généraliste, comme on aurait créé une application Windows Forms.

Vous pouvez également créer un projet de type WPF Browser Application, qui est en fait un projet XBAP. Comme on le verra dans un chapitre ultérieur, XBAP est une application WPF s'exécutant dans un navigateur Web.

On peut créer un projet de type WPF User Control Library qui est tout simplement une bibliothèque de contrôles utilisateurs pour WPF.

Pour finir, on a aussi la possibilité de créer un projet de type WPF Custom Control Library, qui vous permettra d'étendre les fonctionnalités d'un contrôle WPF déjà existant.

Donc ici on sélectionnera WPF Application basique… Donnez un nom à votre Projet et cliquez sur OK, vous devriez arriver sur une interface comme la suivante :



Comme vous pouvez le voir, l'interface que nous procure Visual studio pour du WPF est similaire à celle que l'on aurait obtenu pour un projet WinForms basique, à part la fenêtre dock sur le bas avec le Code XAML dont on a pu parler ci-dessus, va nous permettre de modéliser notre fenêtre.

En effet comme on a aussi pu le voir précédemment en WPF, on sépare le code behind (C#) du code designer(XAML). il est normal que le designer, qui va customiser notre fenêtre, propose le code XAML.

5.2 Création d'une ressource

Tout d'abord, pour faire un petit récapitulatif, une ressource d'une application peut-être une icône, une image, fichiers multimédia etc.

Depuis WPF, il existe un nouveau type de ressource spécifique à WPF. En effet ces ressources sont en fait des objets stockés dans des dictionnaires de ressources. Cela peut être des couleurs, des dégradés…

Pour commencer nous allons utiliser notre projet créé précédemment créé, et nous allons ouvrir dans notre concepteur le fichier Window1.xaml

Ensuite dans notre éditeur XAML de notre fenêtre nous allons entrer le code suivant :

```
<!--XAML-->
<Window x:Class="WPFDotnetFranceCourse.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">

    <Window.Resources>
        <SolidColorBrush x:Key="BackgroundBrush" Color="Azure" />
    </Window.Resources>
</Window>
```

On ne va pas expliciter plus ce code source, il y a énormément de ressources qui existent mais ce qui reste inchangé c'est la création de ces ressources, c'est ce qui est le plus important ici. Il vous suffit de placer votre ressource comme vous pouvez le voir entre `<Window.Resources></Window.Resources>`.

5.2.1 Utilisation d'une ressource Statique

On va maintenant voir comme on va faire pour utiliser ces ressources. Je ne vais pas m'attarder sur ce qu'est une ressource statique pour le moment, vous comprendrez par la suite la différence entre une ressource statique et dynamique.

Vous allez voir que l'utilisation de ces ressources nous rappelle l'utilisation de feuilles de style CSS.

```
<!--XAML-->
<Window x:Class="WPFDotnetFranceCourse.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">

    <Window.Resources>
        <SolidColorBrush x:Key="BackgroundBrush" Color="Azure" />
    </Window.Resources>

    <Button Margin="10" Padding="10" Background="{StaticResource
BackgroundBrush}">
        My Button
    </Button>
</Window>
```

Ainsi, comme on peut l'observer, nous avons juste spécifié à la propriété background de notre bouton, la ressource à utiliser pour la coloration du fond et le tour est joué.

5.2.2 Utilisation d'une ressource Dynamique

La différence qu'il y a entre une ressource WPF dynamique et statique est tout simplement que lorsqu'une ressource est dynamique, elle sera automatiquement chargée à chaque fois qu'elle est modifiée.

Pour ce qui est de la déclaration de la ressource, pas de changement :

```
<!--XAML-->
<Window x:Class="WPFDotnetFranceCourse.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Window.Resources>
        <SolidColorBrush x:Key="BackgroundBrush" Color="Azure" />
    </Window.Resources>

    <Button Margin="10" Padding="10" Background="{DynamicResource
BackgroundBrush}">
        My Button
    </Button>
</Window>
```

Vous l'aurez compris : dans notre exemple cela ne change pas grand chose, il faudrait plutôt se placer dans une optique ou nos ressources sont des images ou du contenu, qui aura tendance à changer durant le cycle de vie de votre application.

6 Conclusion

Dans cette brève introduction, nous avons vu quelques notions élémentaires du WPF. Elles vous seront utiles afin de mieux appréhender les prochaines notions.

Nous allons voir dans le prochain chapitre comment créer nos premières petites applications, en utilisant divers conteneurs et contrôles de base. Veillez donc à installer tous les outils nécessaires au développement WPF avant d'en débuter la lecture.

Pour un complément d'information pour vos débuts à WPF, vous pouvez consulter les documents publiés par Microsoft sur le MSDN : http://msdn.microsoft.com/fr-fr/library/ms742119.aspx.