2

Création d'une application Silverlight

Description de l'application

Au cours de ce chapitre, nous allons créer une application Silverlight qui :

- réalise plusieurs animations, à savoir un texte déroulant et un carrousel ;
- permet de tourner les pages d'un catalogue touristique au moyen de la souris ;
- diffuse en permanence une vidéo au centre de la fenêtre ;
- recherche des images sur le site Flickr (avec critères de recherche spécifiés par l'utilisateur) et les affiche sur un carrousel tournant.

La figure 2-1 représente l'application terminée et visualisée dans Firefox version 3.

Figure 2-1

D'autres éléments auraient pu être ajoutés, tels que des animations Flash (voir la section « Animation Flash dans une page Silverlight » du chapitre 16), qui ont le mérite d'être largement répandues depuis des années, ou encore la technologie Deep Zoom (voir la section « Deep Zoom » du chapitre 7) qui permet de zoomer sur des sujets de plus en plus précis.

Démarrage du projet avec Visual Studio

Pour réaliser l'application souhaitée, nous utiliserons la version française de Visual Studio 2008. À noter qu'avec la version finale de Silverlight, il sera possible d'utiliser Visual Web Developer (y compris en version française) qui est la version gratuite de Visual Studio. En attendant, vous pouvez télécharger et utiliser la version d'évaluation française de Visual Studio 2008 Pro (voir la section « Installation du logiciel » du chapitre 1).

Pour commencer, vous allez devoir créer un nouveau projet dans Visual Studio en spécifiant que vous utiliserez le C#. À noter, vous pourriez tout aussi bien utiliser Visual Basic (VB.NET). En effet, les deux langages présentent les mêmes possibilités et le choix de l'un ou de l'autre est affaire purement personnelle. Ceux qui ne seraient pas familiers avec l'un de ces deux langages, mais sans être pour autant néophytes en programmation, trouveront dans l'annexe tout ce qu'il faut savoir pour débuter la programmation Silverlight 2 dans l'un de ces deux langages. Dans la suite de l'ouvrage, les versions C# et VB seront toujours présentées conjointement.

Pour créer le projet de l'application, sélectionnez le menu Fichier>Nouveau>Projet> Silverlight>Application Silverlight (onglet C# dans notre cas). Indiquez le répertoire dans lequel il sera enregistré dans le champ `Emplacement` et nommez le projet, ici, `NotreApplication` (figure 2-2).

Visual Studio vous demande maintenant (figure 2-3) si vous créez une véritable application qui devrait être déployée sur un serveur (proposition par défaut), ou un petit projet de test.

Le deuxième choix ne se justifie que s'il s'agit vraiment de créer une petite application de test. Vous gagnerez alors de la place sur le disque et un peu (si peu) de temps lors des compilations.

Validez la proposition par défaut (premier choix), qui correspond à un cas pratique avec développement d'une application et, éventuellement à la fin, déploiement de l'application sur un serveur Internet pour mise à disposition du public. De plus, cela vous en apprendra bien plus sur le fonctionnement d'une application Silverlight.

En procédant ainsi, Visual Studio va créer deux répertoires (`NotreApplication` et `NotreApplicationWeb`) et deux projets dans la solution (une solution regroupe un ou plusieurs projets). Dans le répertoire `NotreApplicationWeb`, seront enregistrés les fichiers qu'il faudra copier sur le serveur (chez votre hébergeur si vous ne disposez pas de votre propre serveur Internet).

Figure 2-2

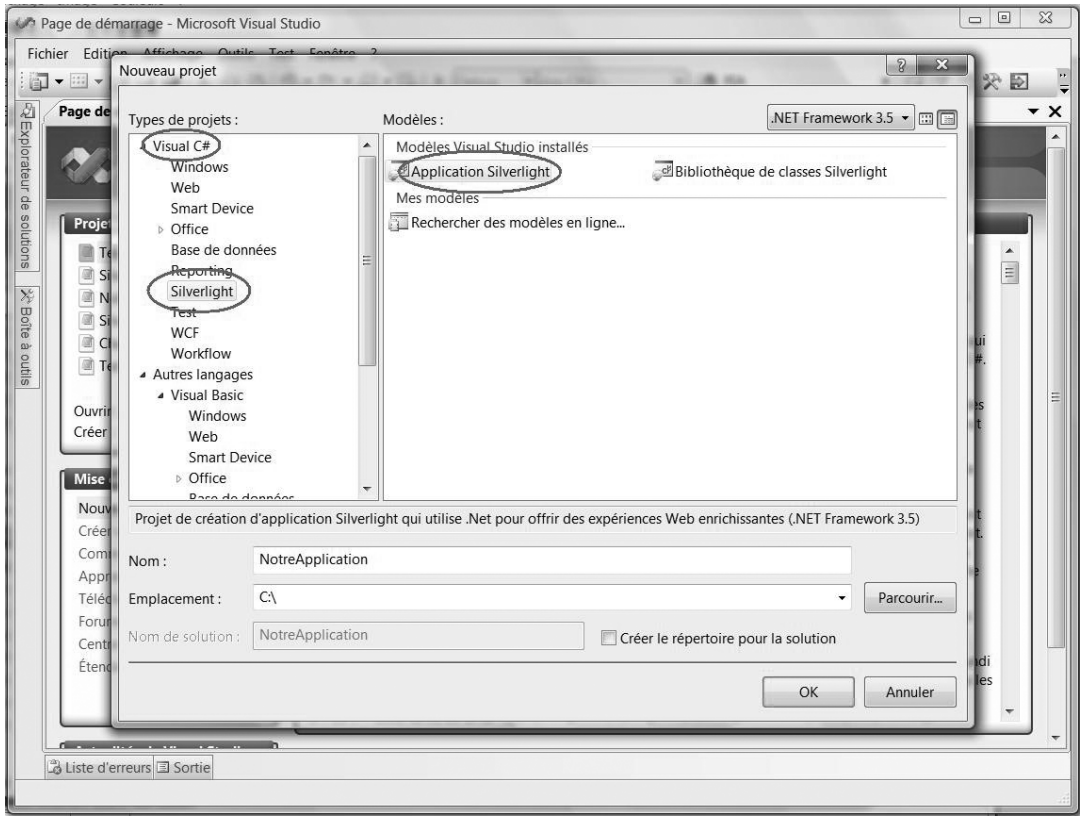
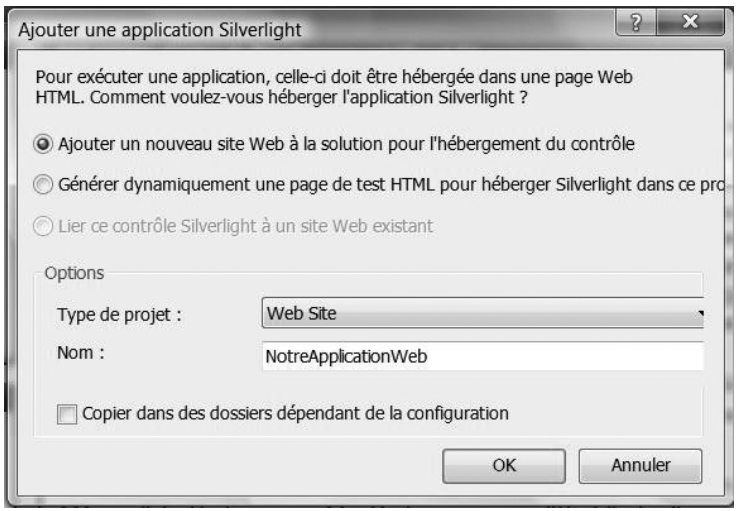


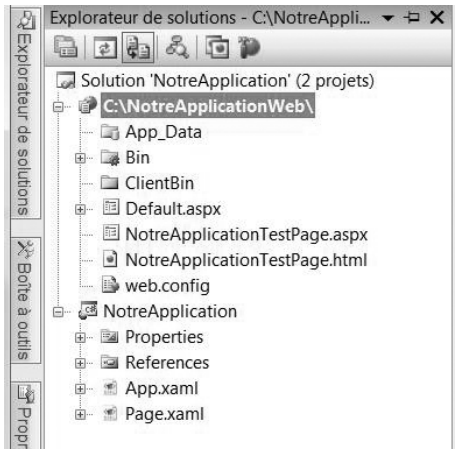
Figure 2-3



Côté hébergeur, aucun traitement spécial n'est à demander. Peu importe que le serveur d'accès à Internet tourne sous IIS (solution Microsoft) ou Apache (solution Linux). Si vos simples pages HTML sont acceptées, vos pages Silverlight le seront aussi.

La figure 2-4 représente l'Explorateur de solutions de Visual Studio tel qu'il apparaît après la génération des squelettes de fichiers de l'application (ces fichiers seront complétés durant la phase de développement de l'application Silverlight).

Figure 2-4



Les fichiers de l'application

Durant le développement de l'application, vous travaillerez essentiellement dans le projet `NotreApplication`, donc dans la partie Silverlight. Une fois l'application terminée, le déploiement (sur le serveur) se fera à partir des fichiers contenus dans `NotreApplicationWeb`, donc dans la partie Web du projet. Après compilation, des fichiers seront automatiquement copiés et mis à jour dans la partie Web du projet.

Les deux fichiers avec lesquels vous travaillerez le plus au cours du développement sont `Page.xaml` et `Page.xaml.cs`, mais examinons au préalable le contenu des différents fichiers générés par Visual Studio.

Visual Studio a créé dans le projet `NotreApplicationWeb` deux fichiers nommés par défaut `NotreApplicationTestPage.html` et `NotreApplicationTestPage.aspx`. Pour renommer ces fichiers à votre guise, cliquez droit sur l'un des noms et choisissez Renommer.

Les fichiers `NotreApplicationTestPage.html` et `NotreApplicationTestPage.aspx` correspondent à des déploiements différents. Pour un déploiement ASP.NET (avec fichier `.aspx`), le serveur doit être sous contrôle d'IIS (*Internet Information Server* de Microsoft). ASP.NET est très utilisé depuis quelques années pour des solutions dites de programmation serveur par les entreprises. Un déploiement HTML convenant parfaitement aux applications Silverlight, nous lui donnerons la préférence en raison de la plus grande facilité de déploiement qu'il

permet (un déploiement HTML est possible sur tous les types de serveurs, ce qui n'est pas le cas pour un déploiement ASP.NET).

Le fichier `Page.xaml` généré par Visual Studio contient la description de la page Web Silverlight. XAML (pour *eXtensible Application Markup Language*) est le « langage » de description de pages exécutées sous contrôle du run-time Silverlight. Il s'agit d'un formalisme XML avec des noms de balises et d'attributs bien particuliers. En général, les programmeurs préfèrent manipuler directement les balises XAML dans Visual Studio, alors que les graphistes préfèrent passer sous Expression Blend, l'outil graphique modifiant ou générant ces balises XAML. Les deux logiciels sont conçus pour un travail en collaboration : toute modification effectuée dans l'un est automatiquement détectée et prise en compte par l'autre. Dans ce chapitre, nous donnerons la préférence à la première possibilité (XAML sous Visual Studio) car vous en apprendrez ainsi bien plus. Tout programmeur Silverlight se doit de maîtriser le XAML (ce qui n'a vraiment rien de compliqué) même s'il lui arrive de passer à Expression Blend pour des opérations plus compliquées.

Au démarrage du développement d'une application, le fichier `Page.xaml` contient le code suivant :

```
<UserControl x:Class="NotreApplication.Page"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="400" Height="300">
 <Grid x:Name="LayoutRoot" Background="White">

 </Grid>
</UserControl>
```

Par défaut, Visual Studio, en générant ces balises, limite la page à un rectangle de 400×300 pixels. Supprimez les attributs `Width` et `Height` pour que la page Silverlight occupe toute la fenêtre du navigateur. Visual Studio a aussi créé une grille comme conteneur de la page Silverlight. Les composants (images, boutons, grilles de données, animations, etc.) seront insérés dans cette grille, ici avec un fond blanc (attribut `Background`). Les conteneurs et en particulier la grille, avec ses nombreuses possibilités, seront étudiés au chapitre 3.

Visual Studio génère également (dans les fichiers `Page.xaml.cs` ou `Page.xaml.vb` selon le langage retenu) un embryon de code avec les `using` (`Imports` en VB) les plus fréquemment utilisés. Il mentionne 11 espaces de noms (il s'agit de noms donnés à des regroupements de classes). C'est dans ce fichier (qui sera complété au fur et à mesure) que vous indiquerez le code à exécuter quand, par exemple, l'utilisateur clique sur un bouton. Pour le moment, il ne contient que le constructeur de la classe `Page`, qui est la classe de la page Silverlight.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace NotreApplication
{
 public partial class Page : UserControl
 {
  public Page()
  {
   InitializeComponent();
  }
 }
}
```

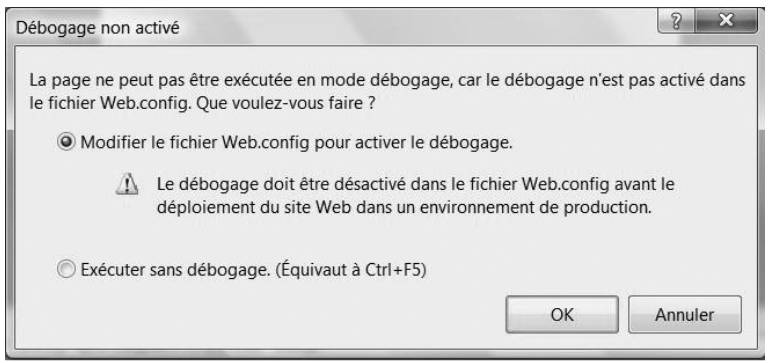
Ceux qui ont déjà pratiqué la programmation .NET pour le développement d'applications Windows se retrouvent en terrain connu avec les mêmes langages, les mêmes outils et quasiment les mêmes classes. Quant aux autres, nul doute qu'ils « entreront » avec une facilité déconcertante dans Visual Studio.

On peut déjà tester cette application, même si elle n'affiche encore qu'une page vierge. Pour lancer l'application, plusieurs techniques sont possibles :

- par le menu Déboguer>Démarrer le débogage, appuyer sur la touche F5 ou encore cliquer sur le petit triangle vert avec pointe vers la droite dans la barre des boutons ;
- par le menu Déboguer>Exécuter sans débogage pour une exécution sans contrôle du debogueur ;
- par l'Explorateur de solutions, cliquez droit sur le fichier `.html` ou `.aspx` puis sur Afficher dans le navigateur ou Naviguer avec… pour faire apparaître un autre navigateur que celui par défaut.

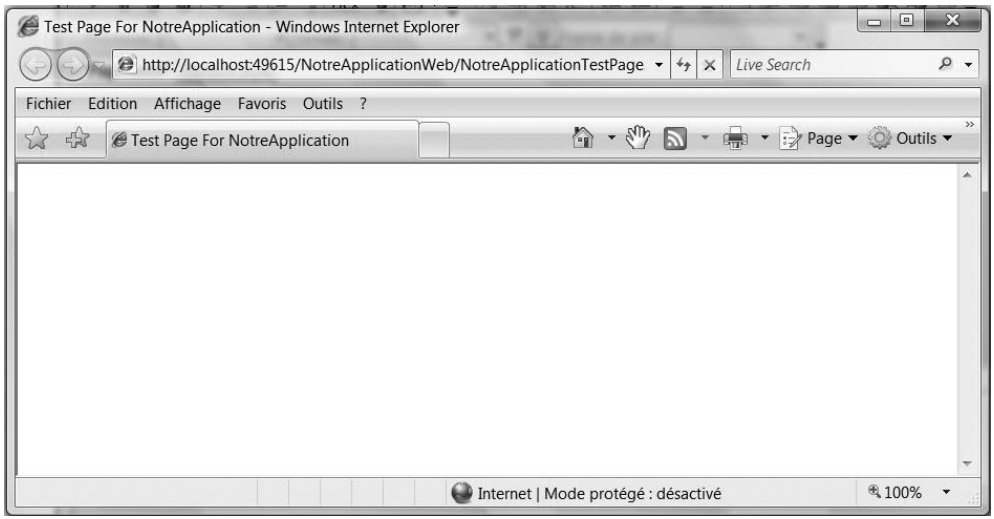
Lors de l'exécution par la touche F5 (en mode débogage, avec possibilité de placer des points d'arrêt), vous devez confirmer lors de la toute première exécution que le débogage de l'application est autorisé (ce qui nécessite une modification du fichier de configuration `Web.config`, figure 2-5).

Figure 2-5



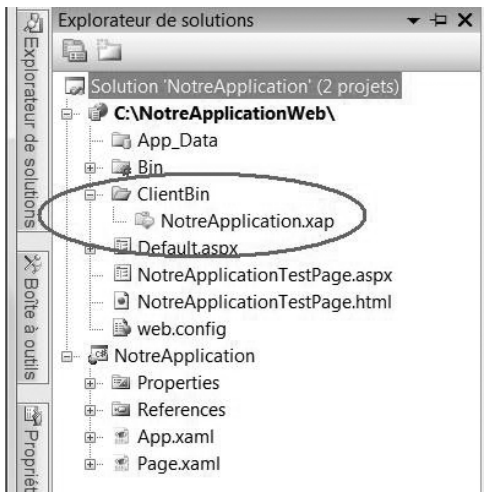
Vous obtenez alors une page Web vierge, ce qui est normal puisque aucun code n'a encore été saisi (figure 2-6).

Figure 2-6



Cette exécution a nécessité la compilation du programme. Un répertoire `ClientBin` a été créé dans la partie Web du projet, lequel contient un fichier d'extension `.xap` (figure 2-7). Ce fichier renferme le code binaire de l'application : il s'agit du résultat de la compilation de l'application (par le compilateur C# ou VB) en ce code binaire qu'on appelle CIL (*Common Intermediate Language*), code qui est indépendant de la plate-forme d'exécution de la page Web (Windows, Mac ou Linux).

Figure 2-7



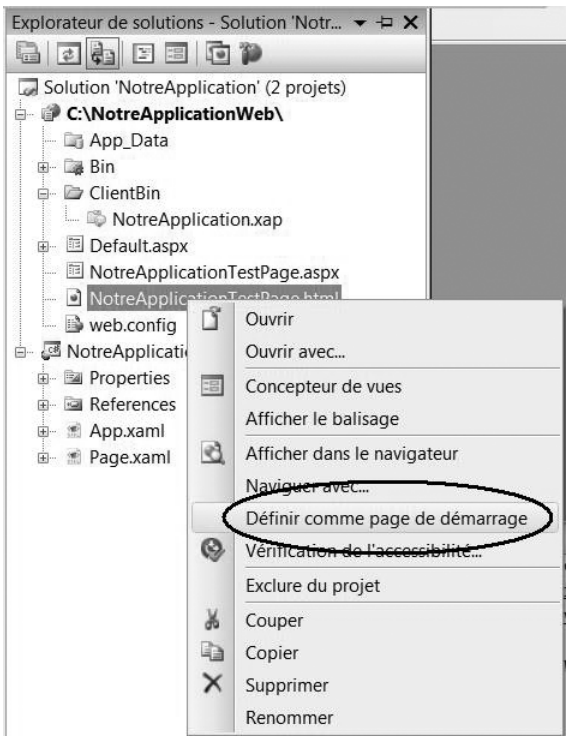
Une application Silverlight 2 est donc bien compilée et ce code est exécuté chez le client par le run-time Silverlight. Comme nous l'avons vu au chapitre 1, l'application détecte au démarrage si le run-time Silverlight est présent. Dans le cas contraire, elle propose à l'utilisateur de l'installer, ce qui prend moins d'une minute et ne réclame aucun droit d'administration de la machine.

Nous verrons bientôt par quel mécanisme le fichier XAP est appelé par le navigateur et exécuté sous contrôle du run-time Silverlight. À noter que le navigateur ne connaît pourtant que le HTML et le JavaScript et n'a pas été modifié pour Silverlight. Nous verrons également à cette occasion que le fichier XAP est en fait un fichier ZIP et qu'il contient, sous forme compressée, différents éléments.

Déploiements ASP.NET et HTML

Passons maintenant à l'examen des fichiers `.aspx` et `.html`. Par défaut, Visual Studio donne la priorité au déploiement ASP.NET, la solution Microsoft de programmation Web côté serveur. Pour modifier ce comportement, cliquez droit sur le nom du fichier HTML (dans la partie Web du projet) et sélectionnez Définir comme page de démarrage (figure 2-8).

Figure 2-8



Lors d'une exécution par la touche F5, c'est désormais le fichier HTML qui est pris en compte par le navigateur. En cours de développement, cette modification n'a d'importance que si vous modifiez un fichier (`.html` ou `.aspx`) sans effectuer la même modification dans l'autre. Pour le déploiement, c'est l'un ou l'autre fichier qu'il faudra copier sur le serveur Internet. Modifier les fichiers `.html` ou `.aspx` n'est vraiment nécessaire que dans des cas particuliers, qui seront présentés au chapitre 16.

Analysons maintenant le fichier `.aspx` généré par Visual Studio (fichier qu'il faudra copier sur le serveur, avec le contenu du répertoire `ClientBin`, en cas de déploiement ASP.NET) :

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Register Assembly="System.Web.Silverlight"
             Namespace="System.Web.UI.SilverlightControls"
             TagPrefix="asp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
 <head runat="server">
  <title>Test Page For NotreApplication</title>
 </head>
 <body style="height:100%;margin:0;">
  <form id="form1" runat="server" style="height:100%;">
   <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
   <div  style="height:100%;">
    <asp:Silverlight ID="Xaml1" runat="server"
                     Source="~/ClientBin/NotreApplication.xap"
                     MinimumVersion="2.0.30523" Width="100%" Height="100%" />
   </div>
  </form>
 </body>
</html>
```

Tout se passe dans la balise `<asp:Silverlight>` qui correspond à un composant écrit par Microsoft dans le cadre d'ASP.NET/Ajax/Silverlight. Facile à utiliser (une seule balise et quelques attributs seulement pour passer sous contrôle de Silverlight) mais malheureusement, cela nous cache le mécanisme de fonctionnement.

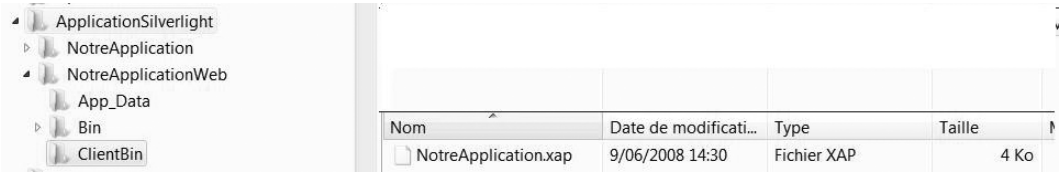
Analysons plutôt le fichier HTML qui nous en apprendra bien plus. Le HTML, tout à fait standard (rappelons que les navigateurs n'ont pas dû être modifiés pour Silverlight), comprend une balise `object`, largement utilisée depuis des années. Cette balise a été conçue à l'origine pour charger « quelque chose » dans le navigateur et demander à du code ainsi téléchargé d'assurer l'affichage de ce « quelque chose » (il s'agissait surtout de permettre la diffusion vidéo dans une page Web). Dans notre cas (voir la balise `param` avec `source`), « quelque chose » doit être téléchargé depuis `ClientBin/NotreApplication.xap` (cet emplacement étant relatif à celui du fichier HTML).

En toute logique, le fichier HTML est appelé en premier par le navigateur (c'est en effet ce fichier qui est mentionné dans l'URL saisie par l'utilisateur). Dès sa réception, ce fichier (de texte) est analysé ligne par ligne par le navigateur. Lorsque celui-ci arrive à la balise `object`, il appelle le fichier `NotreApplication.xap` du répertoire `ClientBin` :

```
<body>
 .....
 <div id="silverlightControlHost">
     <object data="data:application/x-silverlight,"
          type="application/x-silverlight-2-b2" width="100%" height="100%">
    <param name="source" value="ClientBin/NotreApplication.xap"/>
    <param name="onerror" value="onSilverlightError" />
    <param name="background" value="white" />
    <a href="http://go.microsoft.com/fwlink/?LinkID=115261"
       style="text-decoration: none;">
    <img src="http://go.microsoft.com/fwlink/?LinkId=108181"
          alt="Get Microsoft Silverlight" style="border-style: none"/>
    </a>
   </object>
  <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
 </div>
</body>
```

À ce stade du développement de l'application, la taille du fichier XAP est de 4 Ko (figure 2-9).

Figure 2-9



Nous verrons par la suite que la taille de ce fichier augmentera au fur et à mesure du développement de l'application tout en restant étonnamment faible. Ceci vient du fait que ce fichier contient le code compilé de l'application sous forme compressée. Celui-ci est exécuté directement chez le client, sans passer par une interprétation peu efficace, comme c'est le cas avec JavaScript.

Puisque nous en sommes à l'analyse des fichiers `.html` et `.aspx`, profitons-en pour modifier le titre de l'application (balise `title` dans la section `head`) :

```
<title>Une application Silverlight</title>
```

Insertion de l'image de fond

Après ces considérations générales mais indispensables, vous allez désormais pouvoir créer véritablement l'application.

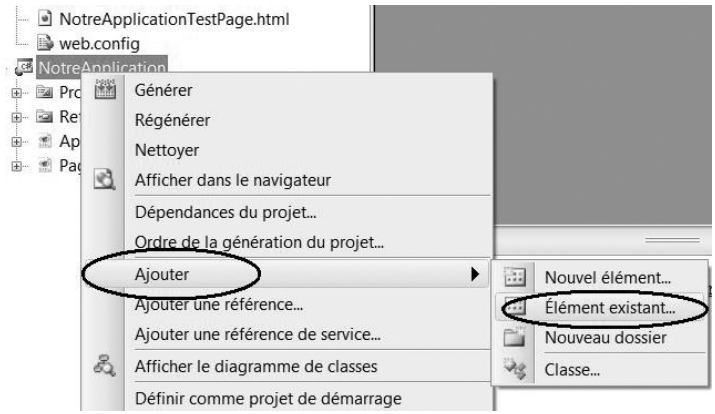
Pour cela, intéressons-nous au plus important des fichiers, à savoir `Page.xaml`. Ouvrez-le en double-cliquant dessus dans l'Explorateur de solutions. Du fait de la suppression précédente des balises `Width` et `Height` dans la balise `UserControl`, le conteneur de l'application Silverlight est une grille qui s'adapte en permanence à la fenêtre du navigateur.

```
<UserControl x:Class="NotreApplication.Page"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   >
 <Grid x:Name="LayoutRoot" Background="White">

 </Grid>
</UserControl>
```

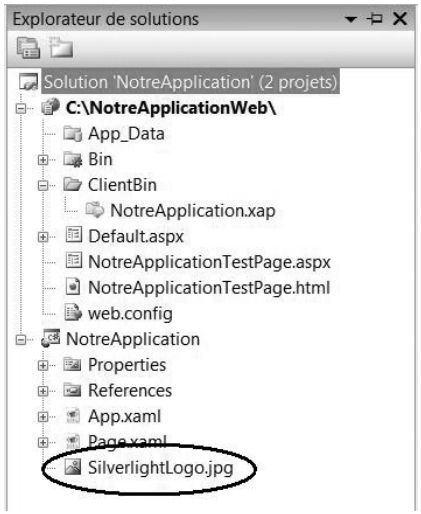
Ajoutons à présent une image (ici, le logo Silverlight) en fond de grille. Pour cela, vous devez charger l'image dans le projet (partie Silverlight) pour qu'elle soit intégrée en ressource de l'application (l'image de fond devra toujours être transmise au navigateur) : cliquez droit sur le nom du projet (partie Silverlight) et sélectionnez Ajouter>Élément existant (figure 2-10).

Figure 2-10



Recherchez ensuite l'image dans le système de fichiers. Elle est copiée dans le répertoire du projet (partie Silverlight) et apparaît dans le projet (figure 2-11) :

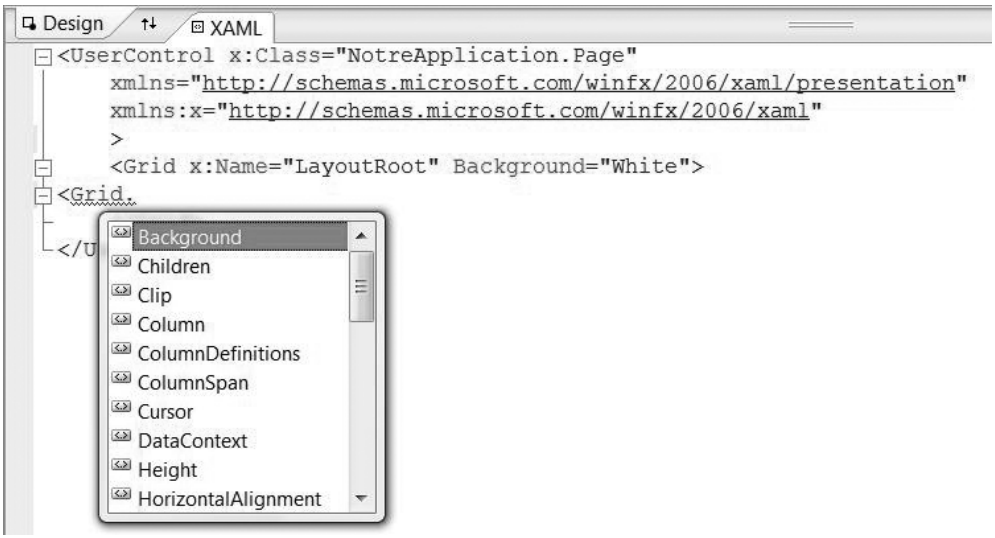
Figure 2-11



Pour le moment, le fond de la grille est blanc mais il va être remplacé par l'image. Pour cela, supprimez l'attribut `Background` et remplacez-le par une balise `Background` (de grille) plus complexe. Il s'agit d'une balise XAML.

Nul besoin de passer des heures à assimiler la syntaxe des balises XAML, l'aide contextuelle de Visual Studio rend les choses très intuitives (figure 2-12).

Figure 2-12



Dans la balise `Grid.Background`, spécifiez un pinceau basé sur une image (les pinceaux seront étudiés au chapitre 4) :

```
<Grid x:Name="LayoutRoot" >
 <Grid.Background>
  <ImageBrush ImageSource="SilverlightLogo.jpg" />
 </Grid.Background>
</Grid>
```

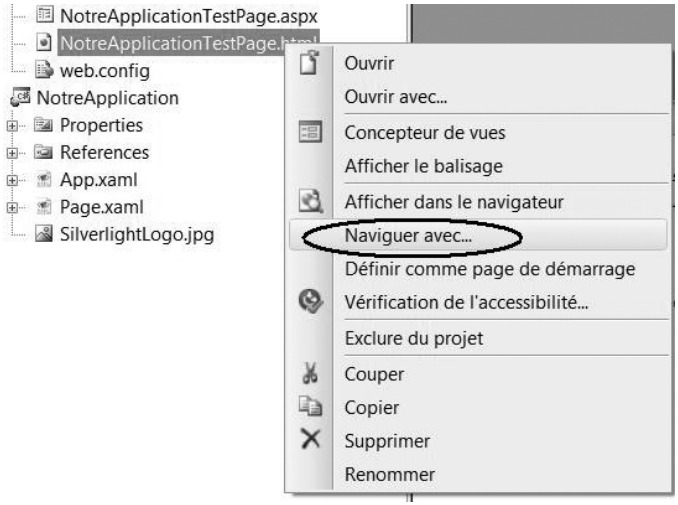
L'application n'a pas encore été compilée mais déjà Visual Studio nous signale une anomalie par un trait ondulé sous `Grid.Background` (figure 2-13). Cette erreur est due au fait que l'attribut `Background` n'a pas été supprimé alors qu'une sous-balise `Grid.Background` a été ajoutée. Supprimez l'attribut. Vous constatez alors que l'anomalie a disparu.

Une image est maintenant affichée en fond de grille mais elle est déformée (elle est, par exemple, aplatie) quand les dimensions de la grille (donc la fenêtre du navigateur) ne correspondent pas (en proportions) à celles de l'image. Pour résoudre ce problème, ajoutez l'attribut `Stretch` avec sa valeur `Uniform` (figure 2-13). Nous reviendrons sur cet attribut au chapitre 7).

Figure 2-13

Exécutez maintenant l'application (ce qui va impliquer une compilation) avec un autre navigateur que celui par défaut. Pour cela, cliquez droit sur le fichier `.html` dans l'Explorateur de solutions (partie Web), choisissez Naviguer avec… (figure 2-14) et sélectionnez Firefox dans la liste des navigateurs disponibles (c'est à vous de créer cette liste et de définir le navigateur par défaut).

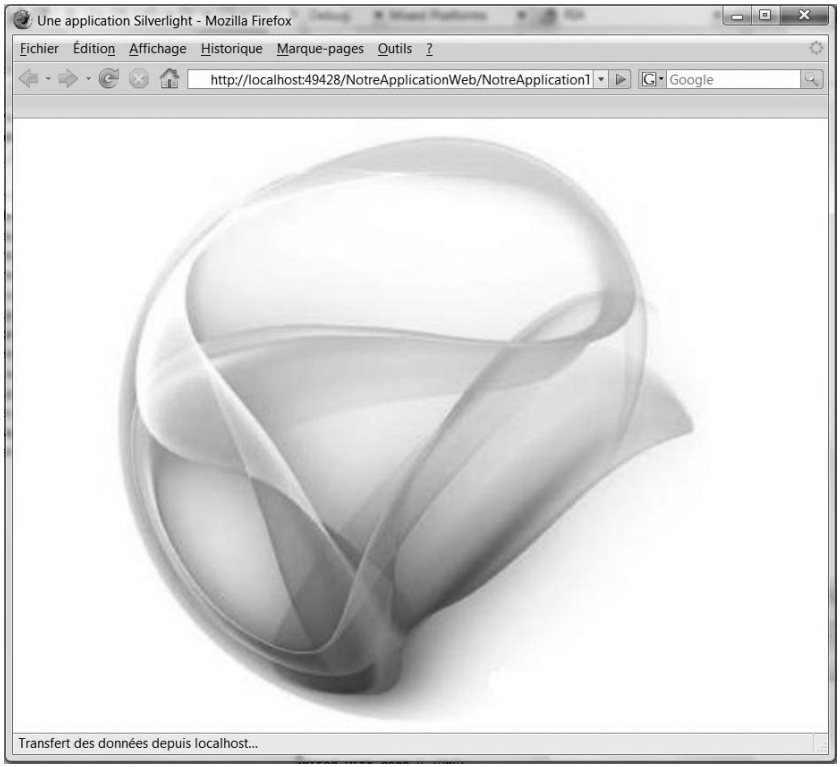
Figure 2-14



La figure 2-15 montre le résultat obtenu dans Firefox.

Quelle que soit la taille de la fenêtre du navigateur, l'image de fond est affichée à sa plus grande taille possible mais toujours en respectant ses proportions. Ainsi, elle n'est jamais déformée.

Figure 2-15



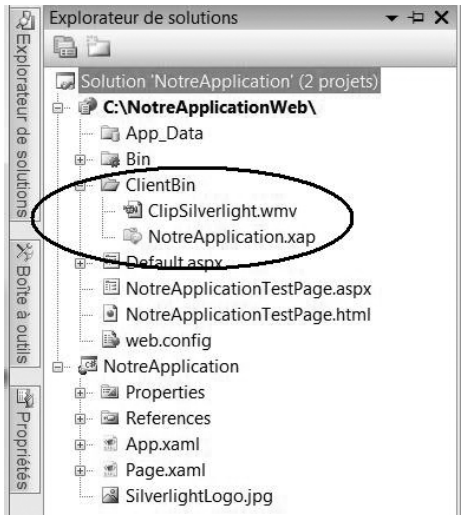
À ce stade, le fichier XAP (qui est un fichier compressé contenant le code binaire de l'application et désormais l'image incorporée en ressource) atteint 17 Ko.

Insertion de la vidéo

Nous allons à présent jouer un clip vidéo (le clip Silverlight de Microsoft) au milieu de la grille, qui se confond avec la fenêtre du navigateur. Pour cela, ajoutez la balise `MediaElement` et indiquez (attribut `AutoPlay`) qu'elle doit être jouée automatiquement et dès que possible (le temps qu'un premier *buffer* soit rempli afin d'assurer par la suite une diffusion fluide du film). Le fichier de la vidéo, `ClipSilverlight.wmv` doit être copié dans le répertoire `ClientBin` (celui du fichier XAP). Ceci s'effectue, par exemple, par un glisser-déposer depuis l'Explorateur de fichiers sur la ligne `ClientBin` de l'Explorateur de solutions, partie Web du projet (figure 2-16).

Insérez maintenant la balise `MediaElement` (voir chapitre 7) qui permet de lire de la musique ou des vidéos. Comme précédemment, vous pouvez procéder par glisser-déposer depuis la barre d'outils de Visual Studio vers un emplacement précis (le point de cliquage) dans le fichier `Page.xaml`. Spécifiez le nom du fichier vidéo (attribut `Source`) ainsi que la taille

Figure 2-16



du rectangle de diffusion (attributs `Width` et `Height`). Attribuez également un nom (attribut `x:Name`) au composant, ce qui n'est pas encore indispensable à ce stade mais le deviendra bientôt :

```
<Grid x:Name="LayoutRoot">
 <Grid.Background>
  <ImageBrush ImageSource="SilverlightLogo.jpg" Stretch="Uniform" />
 </Grid.Background>
 <MediaElement x:Name="video" AutoPlay="True" Source="ClipSilverlight.wmv"
               Width="320" Height="340" />
</Grid>
```

Dans la mesure où les attributs `HorizontalAlignment` et `VerticalAlignment` n'ont pas été spécifiés (voir chapitre 3), `Center` est la valeur par défaut pour ces deux attributs. La vidéo sera donc toujours centrée dans la page du navigateur.

La figure 2-17 illustre le résultat obtenu dans Safari, avec l'image de fond et la vidéo.

La vidéo est maintenant jouée au démarrage (et à chaque rechargement de la page) mais elle ne l'est qu'une seule fois. Il faudra donc détecter la fin de la vidéo et, au moyen d'un programme, la relancer depuis le début. Pour cela, il convient de spécifier l'événement `MediaEnded` qui signale la fin de la vidéo. Il suffit de placer le curseur dans la balise `MediaElement` et d'appuyer sur la barre d'espace pour que Visual Studio affiche la liste des actions possibles (figure 2-18). Il s'agit de tous les attributs et événements applicables à un `MediaElement`, accompagnés d'une courte explication dans une info-bulle bien que les noms des attributs soient en général suffisamment explicites.

Sélectionnez l'attribut `MediaEnded` correspondant à un événement et demandez à Visual Studio de générer automatiquement la fonction de traitement (*event handler* en anglais). Pour cela, saisissez simplement le signe = à droite du nom de l'événement (figure 2-19).

Figure 2-17

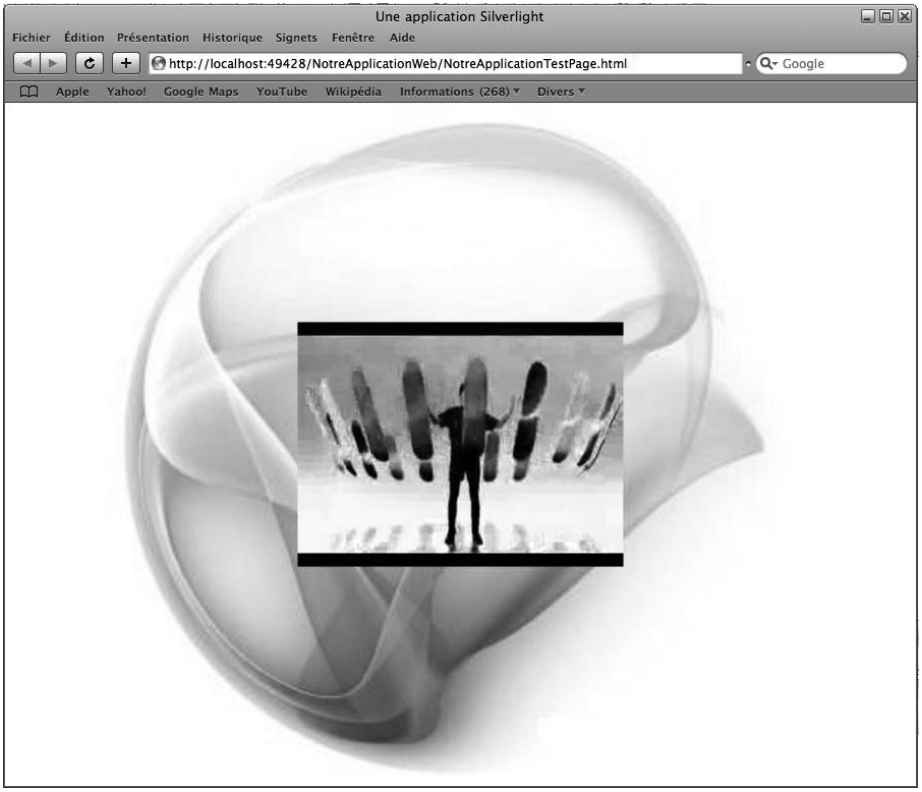


Figure 2-18

```
 <MediaElement x:Name="video" | AutoPlay="True" Source
</Grid>
:/UserControl>
```

Language
LayoutUpdated
Loaded
LostFocus
Margin
MarkerReached
MaxHeight
MaxWidth
MediaEnded
MediaFailed

Figure 2-19

```
<Grid x:Name="LayoutRoot" >
 <Grid.Background>
  <ImageBrush ImageSource="SilverlightLogo.jpg" Stretch="Uniform" />
 </Grid.Background>
 <MediaElement x:Name="video" MediaEnded=""  AutoPlay="True" Source="ClipSilverlight.wmv" Width="320"
</Grid>
</UserControl>
```

<Nouveau gestionnaire d'événements>
Liez les événements à une nouvelle méthode
gestionnaire d'événements' pour naviguer ve

Confirmez ensuite la création de la fonction de traitement au moyen de la touche Tab (il s'agit ici d'une nouvelle fonction de traitement à créer mais on pourrait faire référence à une fonction existante qui traiterait un même événement pour plusieurs composants). La balise devient alors :

```
<MediaElement x:Name="video" MediaEnded="video_MediaEnded" AutoPlay="True"
              Source="ClipSilverlight.wmv" Width="320" Height="340" />
```

tandis que la fonction automatiquement générée pour traiter l'événement est (dans le fichier `Page.xaml.cs`) :

```
private void video_MediaEnded(object sender, RoutedEventArgs e)
{


}
```

Le nom de la fonction de traitement est construit selon le schéma suivant : nom interne de l'élément (attribut `x:Name`), suivi du caractère de soulignement puis du nom de l'événement.

Dans cette fonction, repositionnez la vidéo à 0 seconde et jouez-la avec `Play`. Comme son nom l'indique, la classe `TimeSpan` (littéralement « étendue de temps ») permet de spécifier une durée. Il s'agit d'une classe que l'on retrouve telle quelle (comme la plupart des classes d'ailleurs) en programmation .NET pour Windows ou ASP.NET. Nous avons ici repris la forme la plus simple (ce qui est possible car il s'agit du début de la vidéo) mais on aurait pu construire une durée exprimée en jours, heures, minutes, secondes et millisecondes : `new TimeSpan(jj, hh, mm, ss, milli)`.

Avec ce petit ajout de deux lignes, la fonction de traitement devient :

```
private void video_MediaEnded(object sender, RoutedEventArgs e)
{
 video.Position = new TimeSpan(0);
 video.Play();
}
```

En VB, le XAML est identique et la fonction précédente s'écrit (elle est tout aussi automatiquement générée par Visual Studio) :

```
Private Sub video_MediaEnded(ByVal sender As Object, _
                             ByVal e As RoutedEventArgs)
 video.Position = New TimeSpan(0)
 video.Play()
End Sub
```

Passer d'un langage à l'autre ne présente vraiment aucune difficulté.

Le travail en couches transparentes

Nous allons maintenant nous occuper des animations. Bien que cela ne soit pas strictement nécessaire ici (mais cela nous simplifiera les choses tout en nous permettant de présenter cette intéressante fonctionnalité), nous allons ajouter deux couches transparentes à la grille et réaliser une animation sur chacune d'elles, sans toucher à la grille de fond (`LayoutRoot`) :

```
<Grid x:Name="LayoutRoot" >
 <Grid.Background>
  <ImageBrush ImageSource="SilverlightLogo.jpg" Stretch="Uniform" />
 </Grid.Background>
 <MediaElement x:Name="video" MediaEnded="video_MediaEnded" AutoPlay="True"
               Source="ClipSilverlight.wmv" Width="320" Height="340" />
 <Grid x:Name="GrImagesAnimées" Background="Transparent" >

 </Grid>
 <Grid x:Name="GrLogoAnimé" Background="Transparent" >

 </Grid>
</Grid>
```

Pour commencer, nommez (attribut `x:Name`) chacune des deux grilles transparentes. À ce stade, il s'agit avant tout de les nommer pour les identifier aisément mais les noms donnés aux composants sont aussi importants et jouent le même rôle que les noms donnés aux variables.

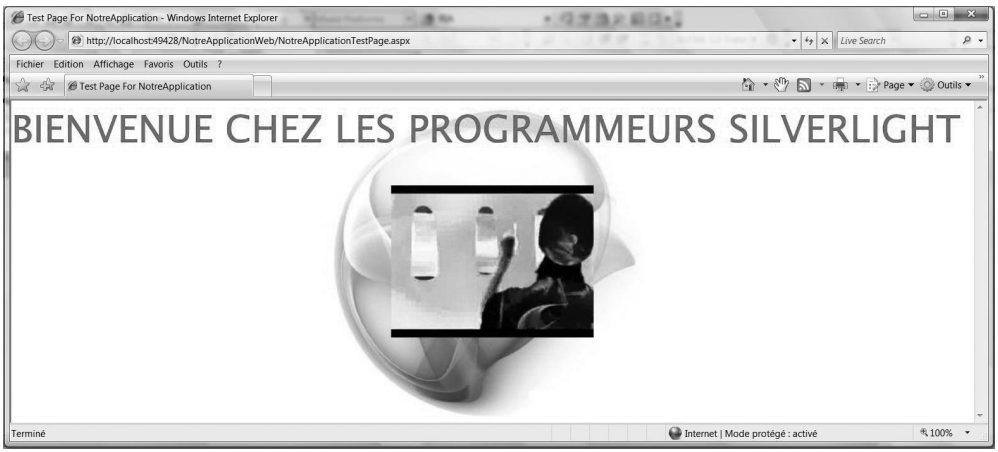
Comme aucune taille n'est spécifiée (ni aucun emplacement relatif), ces deux grilles transparentes se superposent à la grille de fond.

Première animation : le texte déroulant

Dans la seconde grille transparente (`GrLogoAnimé`), ajoutez un texte en rouge et d'une taille de 60 pixels (figure 2-20) :

```
<Grid x:Name="GrLogoAnimé" Background="Transparent" >
 <TextBlock Text="BIENVENUE CHEZ LES PROGRAMMEURS SILVERLIGHT 2"
            FontSize="60" Foreground="Red" />
</Grid>
```

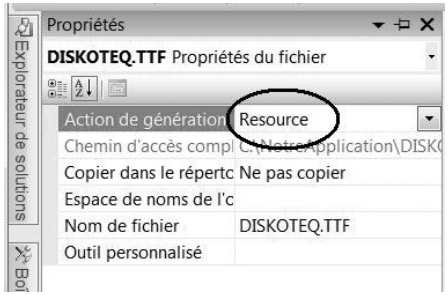
Figure 2-20



Le résultat est assez banal... Nous allons donc télécharger sur Internet une police de caractères (voir la section « Les polices de caractères » du chapitre 5) plus attrayante. Le site `http://www.coolgrafik.com` propose le fichier `Diskoteq.ttf` qui contient la police `Diskoteque`. Comment savoir que `Diskoteq.ttf` contient la police `Diskoteque` ? Dans l'Explorateur de fichiers, il suffit de double-cliquer sur le nom d'un fichier `.ttf` pour afficher les noms des polices de caractères qu'il contient.

Téléchargez ce fichier puis chargez-le dans le projet tout comme vous l'avez fait pour l'image de fond : cliquez droit sur le nom du projet (partie Silverlight), choisissez Ajouter>Élément existant et recherchez le fichier dans le système de fichiers. Vous devez ensuite indiquer que ce fichier doit être incorporé en ressource. Pour cela, cliquez droit sur le nom du fichier `.ttf` dans l'Explorateur de solutions, choisissez Propriétés et sélectionnez la valeur `Resource` dans le champ Action de génération (figure 2-21).

Figure 2-21



Procédez ensuite à un petit changement dans l'attribut `FontFamily` (nom de police) :

```
<TextBlock Text="Bienvenue chez les programmeurs Silverlight 2" FontSize="60"
           FontFamily="Diskoteq.ttf#Diskoteque" Foreground="Red" />
```

afin d'obtenir le résultat de la figure 2-22.

Figure 2-22

La taille du fichier XAP (qui incorpore maintenant la police) est désormais de 34,8 Ko. Comme nous l'avons déjà mentionné, un fichier XAP est en fait un fichier compressé au format ZIP. En l'analysant (il suffit de le renommer en `.zip`), on constate qu'il contient deux fichiers : `AppManifest.xaml` et `NotreApplication.dll`. Le premier donne des informations sur ce qui est transmis au navigateur du client (à l'exception de la page HTML). Le second contient quant à lui le code binaire (résultat d'une compilation) de l'application ainsi que les ressources (dans notre cas et jusqu'à présent, une image et une police de caractères).

Pour notre application, nous décidons de faire défiler le texte en permanence de droite à gauche, comme s'il entrait par la droite et sortait par la gauche de la fenêtre du navigateur. Une animation sera nécessaire pour cela, ce que nous étudierons au chapitre 9.

Pour le moment, placez le texte (balise `TextBlock`) dans un canevas (qui est un type de conteneur, voir chapitre 3), lui-même inséré dans la grille. Spécifiez une marge de 10 pixels par rapport au bord supérieur afin d'aérer la présentation. Dans le canevas, animez la coordonnée `Left` (le long de l'axe horizontal) du texte que vous venez de créer. Cette coordonnée va passer d'un déplacement de 1 400 à –1 400 en 10 secondes et l'animation se répétera sans cesse (attribut `RepeatBehavior`). Il serait possible d'optimiser les valeurs 1 400 et –1 400 en tenant compte de la taille réelle du texte mais cela n'a pas d'importance à ce stade de l'étude.

Pour réaliser l'animation, il convient de placer ce que l'on appelle un *storyboard* dans le jargon Silverlight en ressource de son conteneur (ici, un canevas). Il faut également attribuer un nom (attribut `x:Name`) à l'animation (ici, `animLogo`) ainsi qu'au composant `TextBlock` (ici, `zaLogo`). Ce composant devra par ailleurs être déplacé dans la balise `Canvas`, juste avant la balise de fermeture. Pour plus de détails sur les animations et les transformations, reportez-vous au chapitre 9.

```
<Grid x:Name="GrLogoAnimé" Background="Transparent" >
 <Canvas Margin="0, 10, 0, 0" >
   <Canvas.Resources>
    <Storyboard x:Name="animLogo" RepeatBehavior="Forever" >
     <DoubleAnimation Storyboard.TargetName="zaLogo"
                      Storyboard.TargetProperty="(Canvas.Left)"
                      From="1400" To="-1400" Duration="0:0:10" />
    </Storyboard>
   </Canvas.Resources>
   <TextBlock x:Name="zaLogo"
              Text="Bienvenue chez les programmeurs Silverlight 2" FontSize="60"
              FontFamily="Diskoteq.ttf#Diskoteque" Foreground="Red" />
 </Canvas>
</Grid>
```

L'animation porte sur la propriété `Canvas.Left` de l'élément `zaLogo`, ce qui est spécifié dans les attributs `Storyboard.TargetProperty` et `Storyboard.TargetName`. Du fait que `Canvas.Left` désigne une propriété dite attachée (voir la section « Les rectangles et les ellipses » du chapitre 5), les parenthèses sont nécessaires.

Dans la mesure où il s'agit ici d'animer la propriété `Canvas.Left` qui contient une valeur numérique de type `double` (type privilégié par Silverlight pour les valeurs numériques), nous avons affaire à une animation de type `DoubleAnimation` (d'autres types d'animations seront étudiés à la section « Les animations » du chapitre 9). La durée d'un passage (de droite à gauche dans la fenêtre) est spécifiée dans l'attribut `Duration` et correspond ici à 0 heure, 0 minute et 10 secondes. La durée d'une animation peut aussi être exprimée en millisecondes mais cela n'aurait pas de sens ici.

L'animation sera lancée au démarrage de l'application. L'événement `Loaded` adressé à la grille (voir chapitre 6) signale ce moment. Cet événement est plus précisément signalé juste après le travail de préparation de la page en mémoire et juste avant le tout premier affichage dans la fenêtre du navigateur. Ici aussi, nous demandons à Visual Studio de générer la fonction de traitement. Comme nous l'avons vu précédemment, il suffit de taper `Loaded=` pour que Visual Studio enclenche le processus de création de la fonction de traitement (figure 2-23).

Figure 2-23

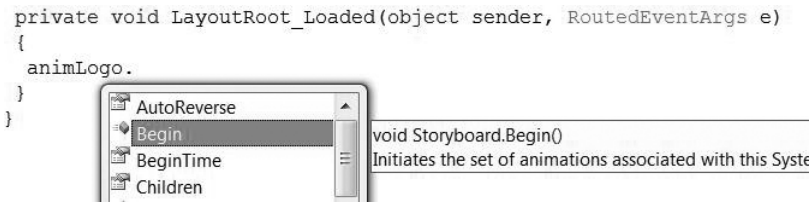
La balise devient alors :

```
<Grid x:Name="LayoutRoot" Loaded="LayoutRoot_Loaded">
```

et une nouvelle fonction de traitement (de l'événement `Loaded` adressé à la grille `LayoutRoot`) est générée dans le fichier `Page.xaml.cs`. Il convient à présent de la compléter. Comme l'animation s'appelle `animLogo` (attribut `x:Name` du storyboard, figure 2-24), la fonction `Begin` appliquée à l'animation la fait démarrer :

```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
 animLogo.Begin();
}
```

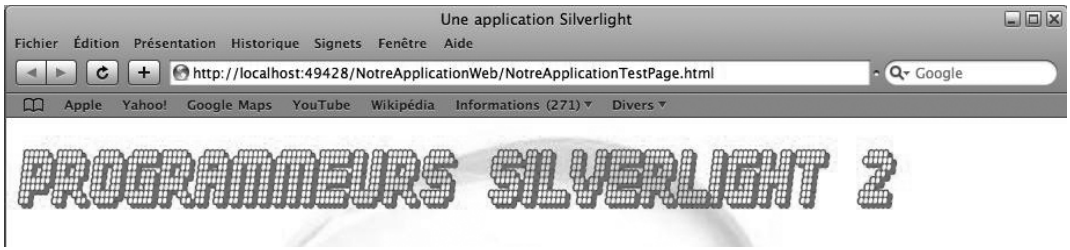
Figure 2-24



Les figures 2-25 et 2-26 présentent le résultat de cette animation dans Safari à deux moments différents.

Figure 2-25

Figure 2-26



Zone d'édition et bouton dans Silverlight

Au bas de la fenêtre, il convient à présent de placer le libellé (`Critère Flickr :`), la zone d'édition (dans laquelle l'utilisateur saisira le critère de recherche) ainsi que le bouton permettant de lancer une recherche d'images sur le site Flickr. Ces trois éléments correspondent respectivement aux composants `TextBlock`, `TextBox` et `Button` qui font partie, avec beaucoup d'autres, de la panoplie des composants Silverlight. Ils devront être accolés au bord inférieur de la fenêtre au moyen de l'attribut `VerticalAlignment` pour lequel la valeur `Bottom` sera spécifiée, tout en prenant soin de définir une marge de manière à aérer la présentation. Ainsi, même si l'utilisateur redimensionne la fenêtre, ces trois éléments seront toujours affichés au bas de la fenêtre.

Pour cela, ajoutez ce qui suit à l'intérieur de la balise `Grid` pour `GrLogoAnimé` (juste avant la balise de fermeture `</Grid>`) :

```
<TextBlock Text="Critère Flickr :" HorizontalAlignment="Left"
           VerticalAlignment="Bottom" Margin="10, 20"
           FontSize="25" Foreground="Red" FontFamily="Verdana" />
<TextBox x:Name="zeCritFlickr" Text ="Paris" HorizontalAlignment="Left"
         VerticalAlignment="Bottom" Margin="200, 25" Width="250" />
<Button x:Name="bFlickr" Content="Recherche Flickr" HorizontalAlignment="Left"
        VerticalAlignment="Bottom" Margin="500, 10" Width="100" Height="60"
        Click="bFlickr_Click" />
```

Profitez-en pour demander à Visual Studio de générer la fonction de traitement du clic sur le bouton (événement `Click`), fonction qui sera complétée par la suite :

```
private void bFlickr_Click(object sender, RoutedEventArgs e)
{

}
```

La figure 2-27 illustre le résultat obtenu à ce stade.

Figure 2-27

Image dans bouton

Pour le moment, le libellé du bouton est un texte (`Recherche Flickr`), que nous allons remplacer par une image, à savoir le logo de Flickr. À l'aide du logiciel Microsoft Photo Editor, ou de tout autre logiciel proposant les mêmes fonctionnalités, rendez ce logo transparent en spécifiant une couleur de transparence et créez le fichier `Flickr.png` à partir du fichier `.jpg`. Incorporez ensuite cette image en ressource (Ajouter>Élément existant dans l'Explorateur de solutions (partie Silverlight). Enfin, modifiez la balise `Button` (supprimez l'attribut `Content` et ajoutez une balise `Button.Content`), qui devient alors :

```
<Button x:Name="bFlickr" HorizontalAlignment="Left" VerticalAlignment="Bottom"
        Margin="500, 10" Width="100" Height="60" Click="bFlickr_Click">
 <Button.Content>
  <Image Source="Flickr.png" />
 </Button.Content>
</Button>
```

La figure 2-28 représente le résultat obtenu, le logo Flickr étant maintenant affiché en transparence dans le bouton.

Figure 2-28

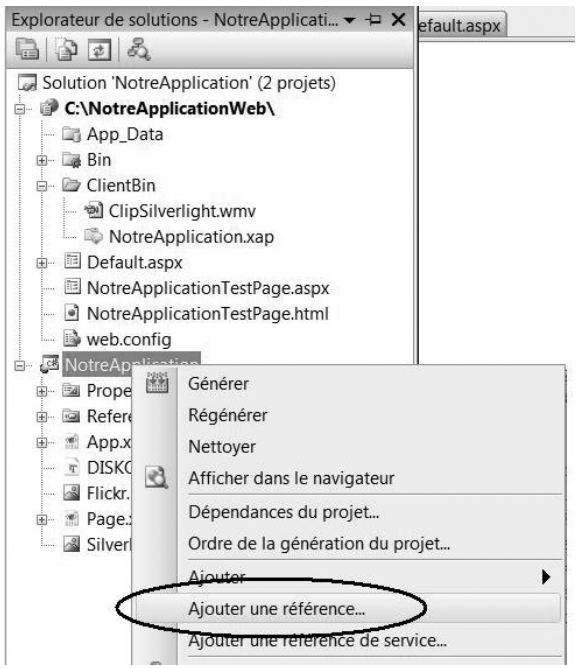
La modification du bouton a été ici très simple. Au chapitre 15, nous verrons comment un graphiste (de préférence…) peut, grâce au logiciel Expression Blend, modifier et personnaliser n'importe quel sous-élément de n'importe quel composant Silverlight (y compris les animations de transition d'état telles que le passage à l'état survol ou encore l'indication du clic) et cela, sans provoquer la moindre modification dans le travail du programmeur.

On tourne la page…

Nous allons maintenant insérer un composant *page turner* dans la partie inférieure droite de l'application, composant qui restera accolé au bord inférieur droit de la fenêtre du navigateur, quelle que soit sa taille. L'utilisateur pourra ainsi tourner les pages d'un catalogue touristique, tout comme il le ferait avec un catalogue papier, mais en se servant bien évidemment ici de la souris. Pour cela, vous allez utiliser le composant décrit à la section « Tourner la page » du chapitre 7.

Pour utiliser ce composant, l'application doit faire référence à (et intégrer) la DLL `SLMitsuControls.dll` (voir cette section « Tourner la page » au chapitre 7). Pour cela, sélectionnez Ajouter une référence… (figure 2-29) dans l'Explorateur de solutions (partie Silverlight).

Figure 2-29



Puis localisez (onglet Parcourir) la DLL `SLMitsuControls.dll` sur votre ordinateur (figure 2-30).

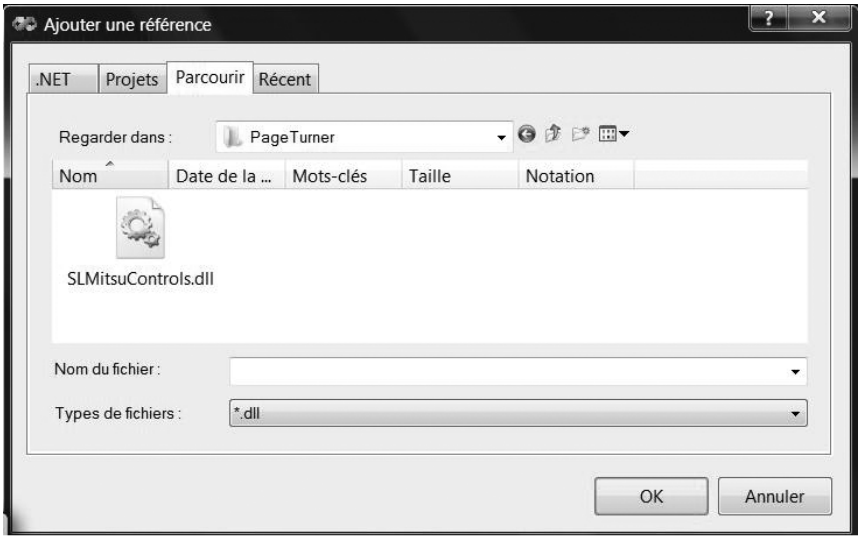
Après compilation, cette DLL sera greffée au fichier XAP, dont la taille augmentera alors de 34 Ko, ce qui est vraiment peu par rapport à ce qu'elle va nous permettre de faire.

Comme cela sera expliqué au chapitre 14, le contrôle utilisateur inclus dans cette DLL doit être spécifié avec un préfixe, de manière à le distinguer des composants intégrés au run-time Silverlight. Pour cela, il suffit d'ajouter la ligne de code suivante (en gras) dans

la balise `UserControl` du fichier `Page.xaml` (le préfixe est ici `mf`, d'après les initiales de l'auteur du composant, à qui nous rendons hommage) :

```
<UserControl x:Class="NotreApplication.Page"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:mf="clr-namespace:SLMitsuControls;assembly=SLMitsuControls"
 >
```

Figure 2-30



Dans la grille `GrLogoAnimé`, ajoutez ce « tourneur de pages » et placez-le (avec une petite marge) dans le coin inférieur droit :

```
<mf:UCBook x:Name="géo" HorizontalAlignment="Right" VerticalAlignment="Bottom"
           Margin="20" Width="500" Height="220" />
```

Par ailleurs, les images du catalogue touristique doivent être copiées dans le répertoire `ClientBin` (partie Web du projet). Il suffit pour cela d'effectuer un copier-coller depuis l'Explorateur de fichiers jusqu'à l'entrée `ClientBin` dans l'Explorateur de solutions (figure 2-31).

Les fichiers de ces images apparaissent alors dans l'Explorateur de solutions, partie Web (figure 2-32).

En ressource dans la grille (celle du fond, `LayoutRoot`), indiquez les pages qui seront utilisées par le « tourneur de pages » :

```
<Grid x:Name="LayoutRoot" Loaded="LayoutRoot_Loaded">
 <Grid.Resources>
  <ItemsControl x:Name="pages" >
   <Image Source="Amsterdam.jpg" Stretch="Fill"  />
```

```
    <Image Source="Venise.jpg"    Stretch="Fill"  />
    <Image Source="Bruxelles.jpg" Stretch="Fill" />
    <Image Source="Moscou.jpg"    Stretch="Fill" />
    <Image Source="Londres.jpg"   Stretch="Fill"  />
  </ItemsControl>
</Grid.Resources>
.....
```

Au chapitre 7, nous verrons qu'il pourrait s'agir de bien autre chose que de simples images. Des événements peuvent être traités, qui indiquent quand les pages sont tournées, quelles pages sont affichées (à gauche et à droite) et sur quelle page l'utilisateur a cliqué, vraisemblablement en vue d'obtenir de plus amples informations.

À ce stade, le fichier `Page.xaml.cs` doit subir quelques modifications. En effet, il faut indiquer que le compilateur doit tenir compte d'un nouvel espace de noms, que la classe `Page` doit implémenter l'interface `IDataProvider` et qu'il faut pour cela implémenter les fonctions `GetCount` et `GetItem` (c'est l'implémentation du composant « tourneur de pages » qui exige ces quelques lignes supplémentaires).

Figure 2-31

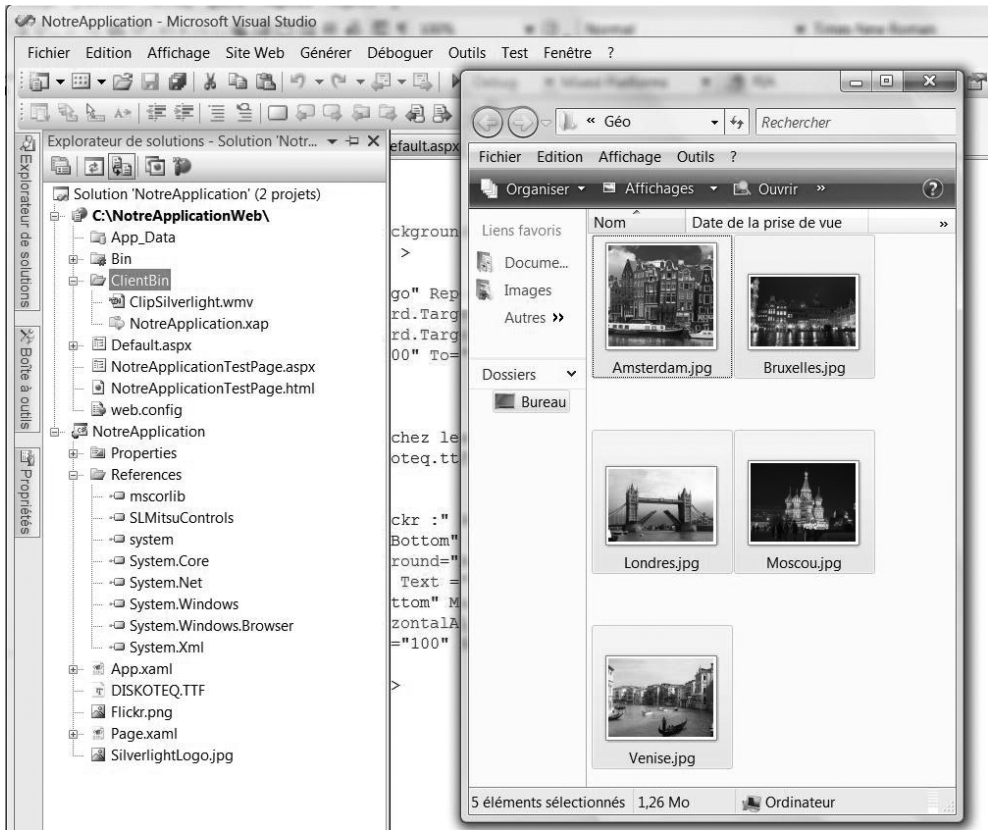
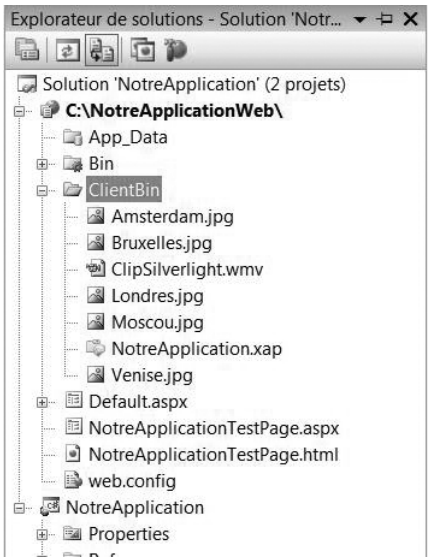


Figure 2-32



Dans la fonction qui traite l'événement `Loaded` adressé à la grille principale (`LayoutRoot`), associez le composant « tourneur de pages » aux données qu'il doit afficher dans ses pages (ici, une série d'images mentionnées dans la balise `ItemsControl`, avec `pages` comme nom interne). Les lignes de code à ajouter dans le fichier `Page.xaml.cs` sont indiquées ci-dessous :

```
using SLMitsuControls;
.....
public partial class Page : UserControl, IDataProvider
{
 .....
 private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
 {
  .....
  géo.SetData(this);
 }
 public object GetItem(int index)
 {
  return pages.Items[index];
 }
 public int GetCount()
 {
  return pages.Items.Count;
 }
```

Vous pouvez désormais lancer l'application et « tourner les pages » (figure 2-33).

Les figures 2-34 et 2-35 présentent le « tourneur de pages » en gros plan, lorsque l'utilisateur tourne la page.

Figure 2-33

Figure 2-34

Figure 2-35

Accès au serveur d'images Flickr

À la demande de l'utilisateur, c'est-à-dire lorsque celui-ci clique sur le bouton, l'application doit aller chercher des images sur le site Flickr et les afficher ensuite sur un carrousel tournant. Dans le cadre de cette application, nous nous limiterons à six photos par sujet (des milliers de photos sont parfois disponibles sur certains sujets).

Pour réaliser cette animation, il convient tout d'abord d'afficher l'anneau sur lequel tourneront les photos. Celui-ci est inséré dans la première grille transparente (`GrImagesAnimées`) :

```
<Grid x:Name="GrImagesAnimées" Background="Transparent" >
 <Ellipse x:Name="rail" StrokeThickness="10" Stroke="Gray"
          Width="500" Height="500" />
</Grid>
```

L'anneau est en fait une ellipse dont la largeur de trait est de 10 pixels. Comme les valeurs par défaut des attributs `HorizontalAlignment` et `VerticalAlignment` sont `Center`, l'anneau sera toujours centré dans la grille (figure 2-36).

Pour obtenir des photos correspondant au critère spécifié dans la zone d'édition, vous devez adresser une requête à Flickr (ceci sera expliqué au chapitre 13, ainsi que la procédure, très simple, d'inscription à Flickr). Après votre inscription sur le site Flickr, celui-ci vous communiquera un code utilisateur (à utiliser ici) ainsi qu'un mot de passe (non utilisé ici car il sert à envoyer des photos au serveur ou donne accès à des photos à usage privé). Cette requête sera exécutée dans la fonction qui traite l'événement `Click` du bouton.

Figure 2-36

La requête doit être adressée à une URL appartenant à Flickr, en mentionnant un code d'accès ainsi que la chaîne « critère de recherche ». Il n'y a pas de véritable problème à communiquer un code d'accès, sauf peut-être un usage immodéré et scabreux que certains pourraient en faire.

```
private void bFlickr_Click(object sender, RoutedEventArgs e)
{
 string clé = "1e1313a40fe8dbe?????24141d90a1231"; // code d'accès
 string url = "http://api.flickr.com/services/rest/"
              + "?method=flickr.photos.search&api_key="
              + clé + "&text=" + zeCritFlickr.Text;
 WebClient client = new WebClient();
 .....   // ici, deux instructions, dont la requête
}
```

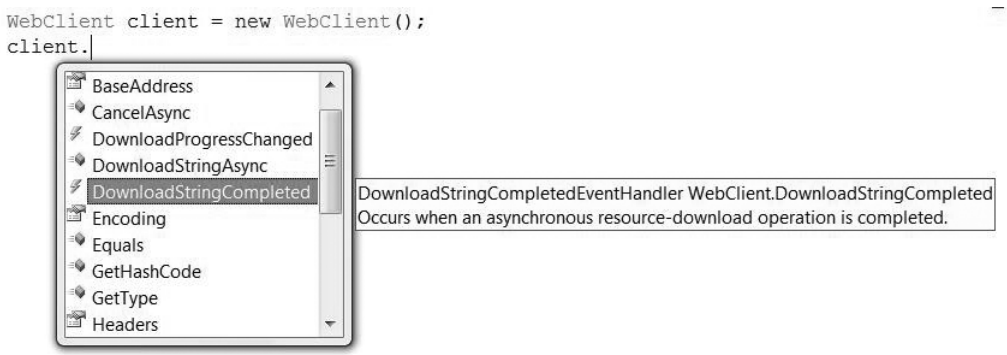
Flickr va lire la requête que vous lui avez envoyée et vous fournira en retour un fichier XML contenant des informations sur les photos répondant au critère (mais pas encore les photos elles-mêmes).

Pour effectuer une telle requête sur le Web, Silverlight met à votre disposition l'objet `WebClient`. Dans la mesure où cette requête peut durer un certain temps, l'opération est

effectuée de manière asynchrone afin de ne pas bloquer l'application Silverlight. L'objet `WebClient` signale l'événement `DownloadStringCompleted` quand le fichier XML a été reçu.

Cet événement est bien sûr repris dans l'aide contextuelle et il convient de signaler à Silverlight qu'il va être traité (figure 2-37).

Figure 2-37

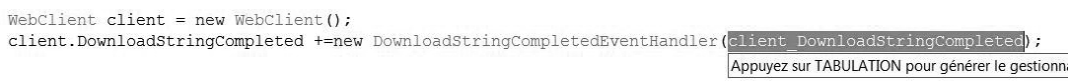


Comme il s'agit d'un événement, saisissez += à droite du nom du nom de l'événement (figure 2-38).

Figure 2-38

Visual Studio vous propose alors de compléter l'instruction et de générer la fonction de traitement (appuyez sur la touche Tab pour valider, figure 2-39).

Figure 2-39



Visual Studio génère donc la ligne de code suivante (avant-dernière instruction de la fonction `bFlickr_Click`) :

```
client.DownloadStringCompleted
   += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
```

ainsi que la fonction de traitement, qu'il convient de compléter :

```
void client_DownloadStringCompleted(object sender,
                                    DownloadStringCompletedEventArgs e)
{
}
```

Dans cette fonction de traitement, `e.Result` contient (dans une chaîne de caractères) la réponse XML.

Ne reste ensuite qu'à lancer l'opération de requête auprès de Flickr (dernière instruction de la fonction `bFlickr_Click`) :

```
client.DownloadStringAsync(new Uri(url));
```

où l'argument est un objet basé sur l'URL où Flickr traite les requêtes (URL qui incorpore dans ses arguments votre code d'accès ainsi que le critère de recherche). Reportez-vous au chapitre 13 pour de plus de détails.

Nous allons maintenant réaliser le carrousel tournant dont la technique sera expliquée en détail au chapitre 9. Pour cela, il convient de définir une transformation de type `RotateTransform` et d'animer l'angle de rotation en le faisant passer de 0 à 360 degrés en 10 secondes et en répétant cette animation à l'infini (sauf pendant les temps de chargement d'une nouvelle série d'images). Le centre de rotation est défini en coordonnées relatives dans l'attribut `RenderTransformOrigin` :

```
<Grid x:Name="GrImagesAnimées"
      Background="Transparent" RenderTransformOrigin="0.5, 0.5" >
 <Grid.RenderTransform >
  <RotateTransform x:Name="rotCarrousel" Angle="0" />
 </Grid.RenderTransform>
 <Grid.Resources>
  <Storyboard x:Name="stbCarrousel" >
   <DoubleAnimation Storyboard.TargetName="rotCarrousel"
       Storyboard.TargetProperty="Angle" From="0" To="360" Duration="0:0:10"
       RepeatBehavior="Forever" />
  </Storyboard>
 </Grid.Resources>
 <Ellipse x:Name="rail" StrokeThickness="10" Stroke="Gray"
          Width="500" Height="500" />
</Grid>
```

Reste désormais à analyser la réponse XML de Flickr, ce que nous verrons en détail au chapitre 13.

Pour décortiquer ce XML de réponse, nous allons utiliser la technologie `Linq for Xml`, le langage d'interrogation et de manipulation de données maintenant intégré à C# et VB (reportez-vous au chapitre 12 pour plus d'informations à ce sujet).

Pour utiliser `Linq for Xml`, une bibliothèque de code doit être intégrée au programme. Pour cela, sélectionnez Ajouter une référence… dans l'Explorateur de solutions, partie Silverlight (figure 2-40). Choisissez ensuite `System.Xml.Linq` dans l'onglet .NET. La DLL de code pour `Linq for Xml` sera ainsi intégrée au projet et finalement greffée dans le fichier XAP envoyé au navigateur (figure 2-41).

Figure 2-40

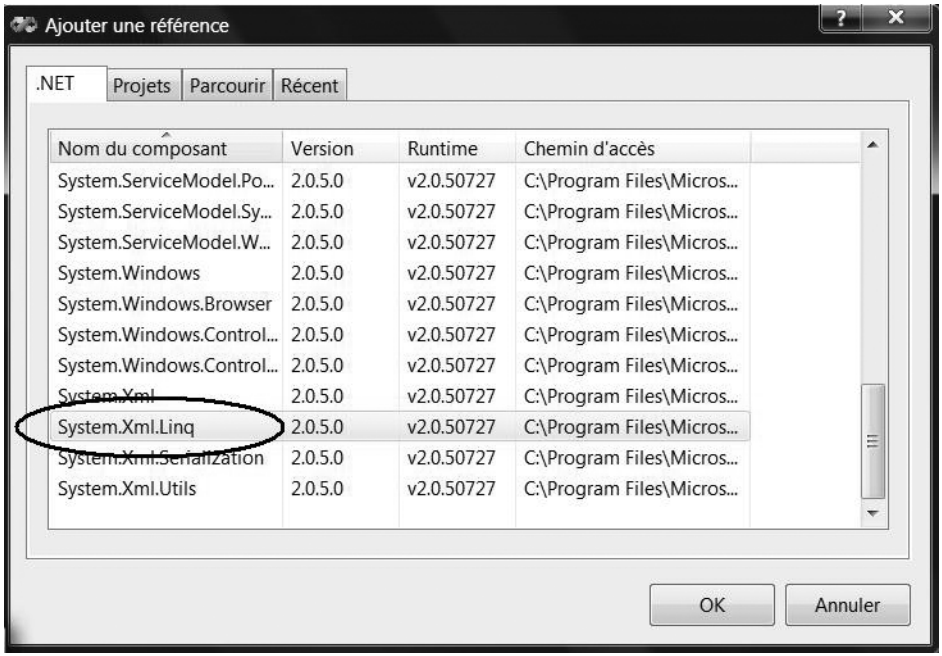
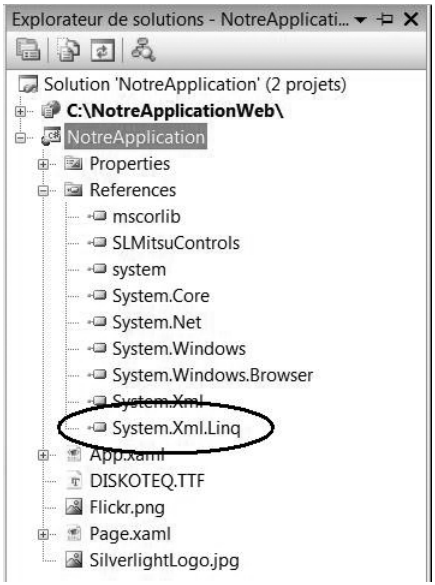


Figure 2-41



Nous allons maintenant décortiquer la réponse XML, créer dynamiquement les composants `Image` (dont on ignore exactement le nombre mais qui sera compris entre zéro et six), télécharger les images depuis Flickr et finalement les accrocher au carrousel tournant. Tout cela sera expliqué en détail aux chapitres 6 (création dynamique de composants), 12 (Linq for XML) et 13 (services Web).

Étant donné les objets utilisés pour réaliser cette animation, il convient d'ajouter les trois espaces de noms suivants en tête du fichier `Page.xaml.cs` :

```
using System.Xml.Linq;
using System.Windows.Markup;
using System.Windows.Media.Imaging;
```

Ce processus d'ajout d'espaces de noms peut être automatisé. Par ailleurs, si Visual Studio ne semble pas connaître une classe ou si une erreur est signalée sur un nom de classe, cliquez droit sur le nom de la classe et sélectionnez Résoudre. Visual Studio procédera alors automatiquement à l'ajout.

Dans la mesure où le nombre d'images qui seront affichées n'est pas connu (ce nombre varie en effet d'une requête à l'autre), il convient de placer les composants `Image` dans une liste dynamique d'images (il s'agit d'une liste qui s'agrandit automatiquement en fonction des insertions). La déclaration et l'instanciation dans la classe est effectuée en dehors des fonctions afin que `listeImages` soit accessible partout :

```
List<Image> listeImages = new List<Image>();
```

Dans la fonction qui traite l'événement `DownloadStringCompleted`, stoppez le carrousel et videz-le (ainsi que `listeImages`) des images qui s'y trouvaient :

```
void client_DownloadStringCompleted(object sender,
                                    DownloadStringCompletedEventArgs e)
{
 stbCarrousel.Pause();

 int N = listeImages.Count;  // nombre actuel d'images
 for (int n = 0; n < N; n++)
 {
  Image img = listeImages[0];
  GrImagesAnimées.Children.Remove(img);
  listeImages.RemoveAt(0);
  img = null;
 }
 .....
```

Décortiquez à présent le XML de réponse en passant par les balises `rsp`, `photos` et `photo`. Cette dernière balise est répétée pour chaque image disponible et contient, dans ses attributs, le titre de la photo ainsi que des informations permettant de reconstituer l'URL de l'image sur le site de Flickr (toutes les opérations sont expliquées en détail dans la suite de l'ouvrage).

```
XDocument xml = XDocument.Parse(e.Result);
var liste = from p in xml.Element("rsp").Element("photos").Elements("photo")
            select p;
liste = liste.Take(6);  // six photos au maximum
N = liste.Count();
```

Pour chaque photo recherchée sur le site de Flickr :

```
for (int n = 0; n < N; n++)
{
 .....
}
```

il faut construire dynamiquement un composant `Image` (ici, à partir d'une balise XAML, voir la section « Création dynamique à partir du XAML » du chapitre 6) :

```
string sXamlImage = "<Image xmlns='http://schemas.microsoft.com/client/2007' "
                    + " Width='200' Height='200' HorizontalAlignment='Left' "
                    + "VerticalAlignment='Top'/>";
Image img = (Image)XamlReader.Load(sXamlImage);
```

Il convient ensuite de préparer l'URL de l'image (sur le site de Flickr) à partir d'attributs de la balise `photo` (voir la section « Application au service Web Flickr » du chapitre 13) :

```
string sFarm = liste.ElementAt(n).Attribute("farm").Value;
string sServer = liste.ElementAt(n).Attribute("server").Value;
string sId = liste.ElementAt(n).Attribute("id").Value;
string sSecret = liste.ElementAt(n).Attribute("secret").Value;
string sUrl = "http://farm" + sFarm + ".static.flickr.com/" + sServer + "/" + sId
              + "_" + sSecret + ".jpg";
```

La position de la photo sur le rail est ensuite calculée (s'il y a X photos, elles font entre elles un angle de 360/X degrés) :

```
double R = rail.Width / 2;
double angle = 2 * 3.14 * n / N;
double X = ActualWidth / 2 + R * Math.Cos(angle) - 75;
double Y = ActualHeight / 2 - R * Math.Sin(angle) - 75;
```

Celle-ci est ensuite positionnée sur le rail :

```
img.Margin = new Thickness(X, Y, 0, 0);
```

puis récupérée sur le site de Flickr :

```
ImageSource imgs = new BitmapImage(new Uri(sUrl));
img.SetValue(Image.SourceProperty, imgs);
```

L'image est alors « accrochée » sur la grille transparente et insérée dans la liste d'images :

```
GrImagesAnimées.Children.Add(img);
listeImages.Add(img);
```

Enfin, le carrousel tournant est lancé :

```
stbCarrousel.Begin();
```

La figure 2-42 montre le résultat final.

Figure 2-42

À noter que l'on pourrait stabiliser les images en leur donnant une rotation inverse à celle de la grille (ce qui sera le cas pour le programme d'accompagnement présenté au chapitre 7).