Hands-On Lab

Windows Phone - Navigation et Contrôles

Version : 1.0.0

Vue d'ensemble

Cet atelier vise à présenter le système de rendu de Windows Phone et le système de navigation, notamment les principes de base de la navigation entre les écrans (pages) d'une application Silverlight pour Windows Phone. Au cours de cet atelier, vous concevrez une application intégrant une navigation entre différents écrans affichant différentes fonctionnalités du téléphone comme l'exécution d'un fichier audio ou vidéo. Vous ajouterez également une barre d'application à la page principale afin d'exposer des fonctionnalités au sein de l'application. Durant les exercices, vous apprendrez à utiliser Microsoft Visual Phone Developer 2010 Express pour concevoir et designer une application Windows Phone. Une connaissance de XAML et de Silverlight 3 sont prérequis pour réaliser cet atelier.

La plateforme applicative Windows Phone ouvre la porte à :

> **Des applications et des jeux fantastiques.** Utiliser les Framework Silverlight et XNA pour concevoir des applications et jeux interactifs et de haute-qualité pour les téléphones Windows Phone 7 Series.
>
> **Des applications rapides et incroyablement riches.** La combinaison de Visual Studio 2010 et Expression Blend avec les spécifications matérielles standards de Windows Phone 7 permet de gagner du temps pour faire de la vision de chacun une réalité sur chaque téléphone.

Plus d'opportunités et de partenariats. Concevez des applications et des jeux innovants avec vos compétences actuelles et les outils disponibles pour Windows Phone 7, et touchez de nombreux clients, indépendamment du matériel qu'ils utilisent. Vous n'avez pas à vous soucier des ventes et de la distribution, la place de marché Windows Phone s'en charge. Codez et recueillez-en les fruits !

Objectifs

A l'issue de cet atelier, vous :

- Aurez une parfaite compréhension du modèle de navigation d'une application Silverlight, et notamment le modèle de navigation au sein des pages et objets à l'intérieur d'une application Windows Phone 7
    - PhoneApplicationFrame
    - PhoneApplicationPage

    - Utilisation des services de navigation
    - Gérer le bouton “Back”
- Saurez contrôler l’orientation du téléphone
- Saurez ajouter une barre d’application dans les pages de votre application
- Connaitrez les contrôles disponibles pour les applications Windows Phone, notamment
    - ListBox
    - Navigateur Web
    - Boutons Hyperlink
    - Media

Prérequis

Sont présentés ci-dessous les prérequis logiques pour réaliser cet atelier :

- Microsoft Visual Studio 2010 Express pour Windows Phone ou Microsoft Visual Studio 2010
- Les outils de développement *Windows Phone Developer Tools*

Exercices

Cet atelier comprend les exercices suivants :

1. Introduction au modèle de navigation Windows Phone
2. Introduction aux contrôles disponibles pour les application Windows Phone
3. Introduction aux services Windows Phone

Durée estimée pour la réalisation de l’atelier: **60 minutes**.

Exercice 1 : Introduction au modèle de navigation Windows Phone

Dans cette section, nous allons ouvrir la solution de démarrage et :

- Ajouter des références de projet
- Ajouter un nom d'application en tant que chaine de caractères partagée
- Editer la page principale et ajouter des dossiers à la solution pour gérer les pages

Nous utiliserons l'environnement de développement Visual Phone Developer 2010 Express et déploierons vers l'émulateur Windows Phone pour le débogage. La solution sur laquelle nous travaillerons est basée sur le modèle d'application Silverlight pour Windows Phone. Au cours du développement, nous ajouterons un élément spécifique au projet Silverlight pour Windows Phone: la "Windows Phone Portrait Page".

Il y a actuellement 2 références par défaut de l'application Windows Phone : **Microsoft.Phone.Controls** et **Microsoft.Phone.Controls.Navigation.** Nous ajouterons 3 références supplémentaires pour gérer notre projet :

- **Microsoft.Phone.Controls.WebBrowser**
- **Microsoft.Phone.Shell**
- **Microsoft.Phone.Tasks**

Le fichier MainPage.xaml, comme tous les éléments **PhoneApplicationPage** créés à partir du modèle d'élément du projet, contient 2 grilles nommées TitleGrid et ContentGrid. Notez également que les contrôles de page gèrent les ressources de style du téléphone définies dans App.xaml.

> **Remarque: Les** étapes de cet atelier sont basées sur l'utilisation de Microsoft Visual Phone Developer 2010 Express, mais sont également applicables à Microsoft Visual Studio 2010 avec les outils de développement Windows Phone Developer Tools. Globalement, les instructions faisant référence à Visual Studio s'appliquent aux 2 produits.

La section suivante fournit quelques informations à propos de la navigation Silverlight et le data binding. Si vous êtes déjà familiers avec ces concepts, vous pouvez directement passer à la Tâche #1.

> **Remarque :** Ci-dessous une présentation générale de la navigation Silverlight

Silverlight pour Windows Phone fournit les classes **PhoneApplicationFrame** et **PhoneApplicationPage** pour faciliter la navigation entre les différentes sections de contenu. Les contrôles **PhoneApplicationPage** représentent des sections de contenu. **PhoneApplicationFrame** agit comme un conteneur pour les contrôles de page et facilite la navigation entre les pages. Vous créez autant de pages différentes que nécessaire pour présenter le contenu de votre application et naviguer entre ces pages. Vous pouvez également activer des URIs conviviales en mappant un pattern URI spécifique à un fichier gérant la requête pour ce pattern. Pour créer des URIs conviviales au sein d'une *frame*, vous pouvez spécifier qu'un pattern URI spécifique est mappé avec une page particulière.

Le mappage d'URI permet de créer une URI décrivant l'action d'un utilisateur plutôt que le chemin vers un fichier.

Un mappage d'URI peut inclure des segments d'espace réservés dans l'URI qui correspondent à une valeur quelconque dans ce segment. Vous spécifiez un segment d'espace réservé en plaçant le nom du segment avec des accolades ({and}). Le segment d'espace réservé agit comme une variable. La demande d'URI est mappée sur le premier motif qui correspond à la demande. Vous pouvez utiliser le contrôle HyperlinkButton pour permettre aux utilisateurs d'accéder aux pages de l'application, en définissant la propriété NavigateUri à une URI qui correspond à une page. En cliquant sur ce contrôle, le Frame navigue vers la page demandée. La page cible peut alors accéder aux paramètres, récupérés grâce à la propriété NavigationContext.QueryString.

Le mappage d'URI peut être utilisé pour fournir des URL plus conviviale, masquer l'emplacement physique de la vue, et offrir une vue unique pour différents paramètres de navigation (comme les chaînes de requête en HTML). Dans l'exemple suivant (non lié directement à l'atelier), vous pouvez voir comment «cacher» l'emplacement physique de la vue ("Home", "A propos"), comment mettre en œuvre la logique de navigation basé sur la valeur prévue ("Page / {number}"), et comment passer des paramètres à la page qui pourrait exécuter une logique métier différente en fonction de leur valeur ("Customers{id}", "Products/{id}", "Orders{type}&{id}&{date}") :

XAML

```
<navcore:UriMapper x:Key="TheMapper">
    <navcore:UriMapping Uri="Home" MappedUri="/Pages/Views/Default.xaml"/>
    <navcore:UriMapping Uri="About-Us"
MappedUri="/Pages/Views/Misc/About.xaml"/>
    <navcore:UriMapping Uri="Page/{number}"
MappedUri="/Pages/Views/Books/Page{number}.xaml"/>
    <navcore:UriMapping Uri="Customers/{id}"
MappedUri="/Pages/Views/Data/Page.xaml?action=getCustomer&amp;id={id}"/>
    <navcore:UriMapping Uri="Products/{id}"
MappedUri="/Pages/Views/Data/Page.xaml?action=getProduct&amp;id={id}"/>
```

```
    <navcore:UriMapping Uri="Orders/{type}&amp;{id}&amp;{date}"
MappedUri="/Pages/Views/Data/Page.xaml?action={type}&amp;orderId={id}&amp;orde
rDate={number}"/>
</navcore:UriMapper>
```

Remarque : Ci-dessous une présentation générale du Data Binding

Le **Data binding** permet aux applications Silverlight de simplement afficher et interagir avec des données. L'affichage des données est séparé de la gestion de celles-ci. Une connexion (ou *binding*) entre l'interface utilisateur et un objet *data* permet aux données de naviguer entre les 2. Lorsque les données changent alors qu'un *binding* est établi, les éléments de l'interface graphique liés aux données peuvent refléter les changements automatiquement. De la même manière, les changements faits par un utilisateur à un élément de l'interface utilisateur peuvent être reflétés dans l'objet *data*. Par exemple, si l'utilisateur édite une valeur dans une TextBox, la valeur de la donnée sous-jacente est automatiquement mise à jour pour prendre en compte cette modification.

D'autres cas usuels de *binding* peuvent être de *binder* une ListBox à une liste de titres, un masque de saisie à un objet *data* côté client, ou encore une Image à une photo de l'utilisateur.

Tout binding doit spécifier une source et une cible. Le schéma ci-dessous présente les concepts de base du binding.

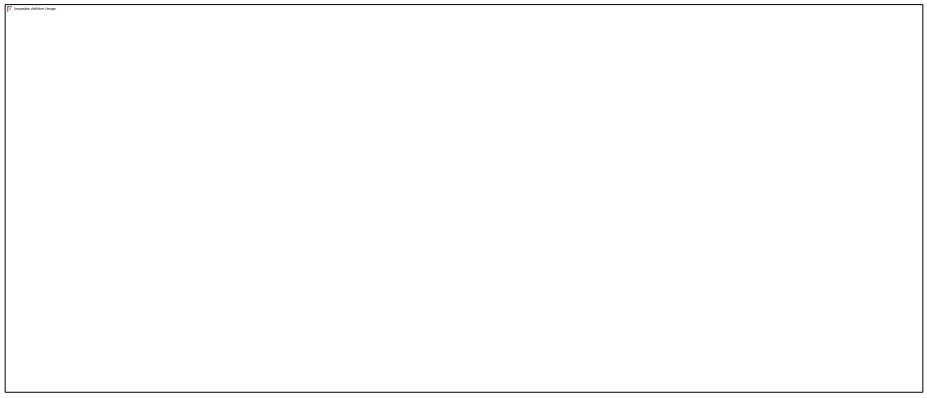

Figure 1

Data binding

Le moteur de binding récupère l'information de l'objet Binding concernant :

- La propriété de l'UI cible affichant les données et permettant à l'utilisateur de les modifier. La cible peut être n'importe quelle **DependencyProperty** d'un **FrameworkElement**.
- L'objet source contenant les données circulant entre la source et la cible. La source peut être tout objet CLR, incluant l'élément cible en tant que tel ou d'autres éléments de l'UI. Si la cible est dans un modèle de données, la source peut être l'élément de l'UI auquel le modèle est appliqué.
- La direction du flux de données. La direction est spécifiée en déterminant la propriété **Mode** sur l'objet **Binding**.
- Le convertisseur de valeur s'appliquant aux données passées. Le convertisseur de valeur est une classe implémentant **IValueConverter**.

Par exemple, la propriété **Foreground** d'une **TextBox** peut être liée à un **SolidColorBrush** de telle sorte que la couleur du texte puisse changer en fonction des données. Dans ce scénario, la propriété **Foreground** est la cible, et l'objet **SolidColorBrush** est la source pour le binding.

Un modèle de données permet de spécifier comment les données sont affichées. Par exemple, votre objet pourrait contenir une image et une chaîne de caractère. Vous pouvez définir le modèle de données pour afficher la chaine de caractère à droite de l'image, à gauche, ou directement sur l'image. Avec des modèles de données, vous pouvez ajouter des marges entre l'image et le texte, une bordure, une couleur de fond. De plus, vous pouvez appliquer le même modèle de données à tous les objets de votre collection en utilisant un **ItemsControl** (comme une **ListBox**).

Tâche 1 – Créer un projet d'application Windows Phone dans Visual Studio

Au cours de cette tâche vous créerez une nouvelle application Windows Phone et explorerez sa structure.

> **Remarque :** Si vous êtes déjà familier avec la création de nouveaux projets avec Visual Studio, ou avez déjà réalisé un autre atelier Windows Phone, vous pouvez créer un nouveau projet d'application Windows Phone de type **WindowsPhoneNavigationAndControls** et vous rendre directement à l'étape 10.

1. Ouvrez Microsoft Visual Studio 2010 Express pour Windows Phone depuis le menu **Start | All Programs | Microsoft Visual Studio 2010 Express**.

   Visual Studio 2010: Ouvrez Visual Studio 2010 depuis le menu **Start | All Programs | Microsoft Visual Studio 2010**.

2. Dans le menu **File**, sélectionnez **New Project**.

   Visual Studio 2010: Dans le menu **File**, allez sur **New** puis sélectionner **Project**.

3. Sélectionnez la catégorie **Silverlight for Windows Phone** dans la liste des modèles installés de la boîte de dialogue **New Project**, puis saisissez le nom « **WindowsPhoneNavigation** » et cliquez sur **OK**.

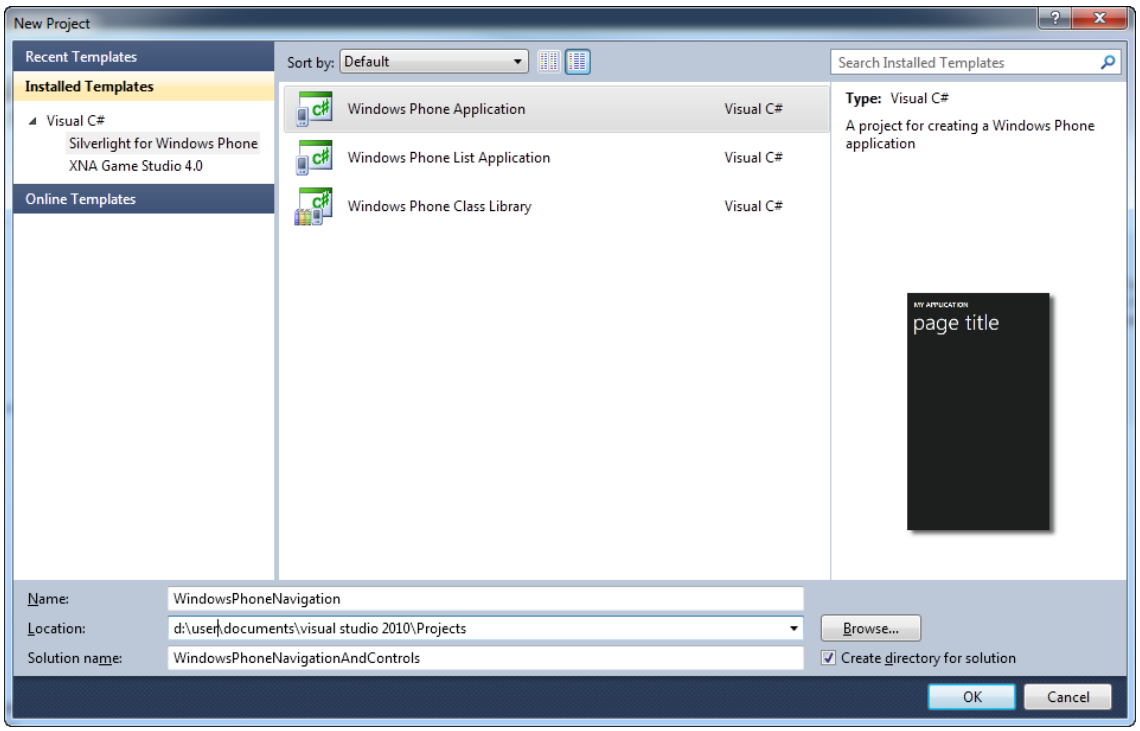

Figure 1

Créer un nouveau projet Windows Phone dans Microsoft Visual Studio 2010 Express pour Windows Phone

4. Dans l'explorateur de solutions, regardez la structure de la solution générée par le modèle d'application Windows Phone. Toute solution Visual Studio regroupe un

ensemble de projets liés ; dans le cas présent, la solution ne comprend qu'un projet Silverlight pour Windows Phone nommé « **WindowsPhoneNavigation** ».

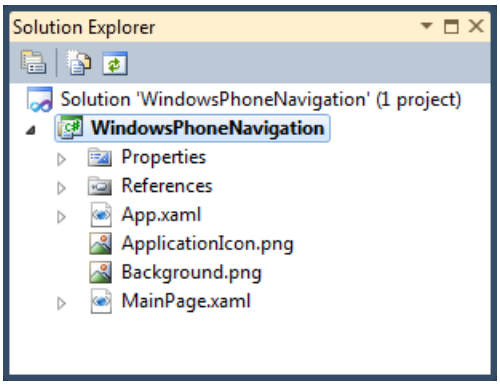

Figure 2

L'explorateur de solutions présentant la structure de l'application WindowsPhoneNavigation

Remarque : l'explorateur de solution permet de visualiser ses éléments et réaliser des tâches d'administration de ceux-ci au sein d'une solution ou d'un projet. Pour afficher l'explorateur de solution, appuyez sur **CTRL + W, S** ou sélectionner **Other Windows | Solution Explorer** dans le menu **View.**

Le projet **WindowsPhoneNavigation** contient les éléments suivants :

Elément	Description
App.xaml / App.xaml.cs	Définit le point d'entrée de l'application, initialise les ressources de l'application, et affiche l'interface utilisateur
MainPage.xaml / MainPage.xaml.cs	Définit une page avec l'interface utilisateur de l'application
ApplicationIcon.png	Un fichier image représentant l'icône de l'application en cours dans la liste des applications du téléphone
Background.png	Un fichier image représentant l'icône de l'application dans le menu de démarrage
Properties\AppManifest.xml	Un fichier de manifeste de l'application requis pour générer la package de celle-ci
Properties\AssemblyInfo.cs	Contient le nom et la version des métadonnées embarqués dans l'*assembly* générée

Properties\WMAppManifest.xml	Un fichier manifest comprenant les métadonnées spécifiques à l'application Silverlight, incluant des fonctionnalités disponibles uniquement avec Silverlight pour Windows Phone
Dossier Références	Une liste de librairies (*assemblies*) fournissant des services et fonctionnalités nécessaires à l'application pour fonctionner

5. Tout d'abord, faites un clic-droit sur **App.xaml** dans l'explorateur de solution, et sélectionnez **View Designer**. Notez que le fichier contient du code XAML avec un élément racine **Application** et en son sein une section **Application.Resources** contenant la définition des couleurs, pinceaux (*brushes*) et styles. Le code XAML initialise la propriété **RootVisual** de l'**Application** pour définir la première page de celle-ci.

App.xaml | MainPage.xaml

```
<Application
    x:Class="WindowsPhoneNavigation.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:system="clr-namespace:System;assembly=mscorlib"
    xmlns:mpc="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls"
    xmlns:phoneNavigation="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Navigation">

    <!--RootFrame points to and loads the first page of your application-->
    <Application.RootVisual>
        <phoneNavigation:PhoneApplicationFrame x:Name="RootFrame" Source="/MainPage.xaml"/>
    </Application.RootVisual>

    <!-- Resources for following the Windows Phone design guidelines -->
    <Application.Resources>
        <!--************ THEME RESOURCES ************-->
        <!-- Color Resources -->
        <Color x:Key="PhoneBackgroundColor">#FF1F1F1F</Color>
        <Color x:Key="PhoneContrastForegroundColor">Black</Color>
        <Color x:Key="PhoneForegroundColor">White</Color>
        <Color x:Key="PhoneInactiveColor">#FF666666</Color>
        <Color x:Key="PhoneDisabledColor">#FF808080</Color>
        <Color x:Key="PhoneSubtleColor">#FF999999</Color>
        <Color x:Key="PhoneContrastBackgroundColor">#FFFFFFFF</Color>
        <Color x:Key="PhoneTextBoxColor">#FFBFBFBF</Color>
        <Color x:Key="PhoneBorderColor">#FFCCCCCC</Color>
        <Color x:Key="PhoneTextSelectionColor">Black</Color>
        <Color x:Key="PhoneAccentColor">#FF1BA1E2</Color>
```

Figure 3

Fichier App.xaml par défaut généré par le modèle d'application Windows Phone

Remarque : le fichier **App.xaml**, combiné avec le code sous-jacent du fichier **App.xaml.cs**, définit une instance de la classe **Application**. Cette classe encapsule une application Silverlight pour Windows Phone et lui fournit son point d'entrée. Elle contient également des ressources applicatives comme les couleurs, pinceaux (*brush*) et objets de style utilisés au travers de l'application. Vous pouvez totalement modifier l'apparence de l'application en changeant les définitions contenues dans ce fichier, comme les feuilles de style contrôlent l'apparence de pages HTML.

La propriété **RootVisual** dans la classe **Application** identifie la page de démarrage de l'application. Toute application Windows Phone dispose d'un unique élément maître dont le type est **PhoneApplicationFrame**. Cet élément intègre un ou plusieurs éléments de type **PhoneApplicationPage** présentant le contenu de l'application, et gérant la navigation entre les pages.

6. Désormais, faites un clic-droit sur **App.xaml** dans l'explorateur de solution et sélectionnez **View Code** afin d'ouvrir le fichier de code sous-jacent. Notez que cette classe dérivée d'**Application** souscrit déjà des gestionnaires de différents évènements de l'application et que le modèle a déjà généré des méthodes que vous pouvez mettre à jour pour exécuter du code au cours du démarrage et l'arrêt de l'application.

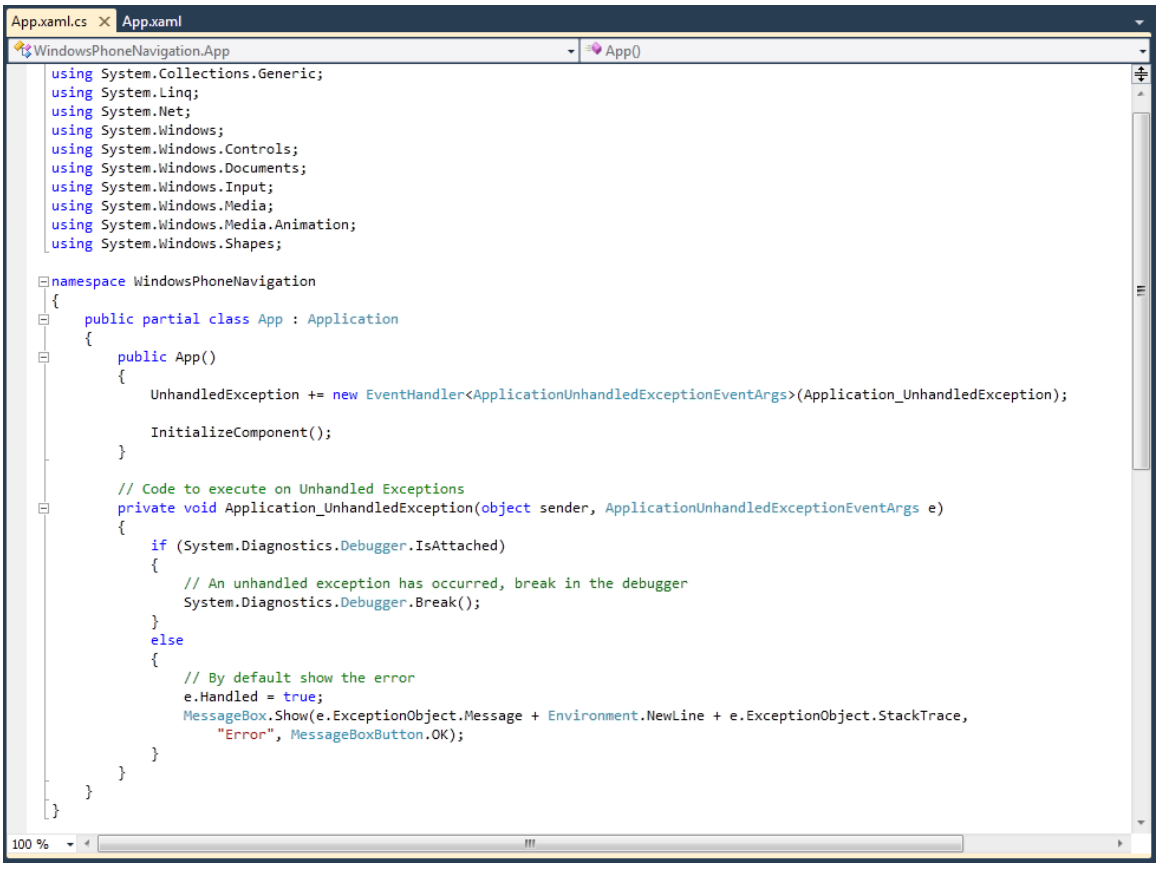

Figure 1

Fichier du code de l'application montrant les gestionnaires d'évènements

Remarque : Même si la classe générée par le modèle d'application Silverlight pour Windows Phone ne comprend pas de gestionnaire pour l'événement

UnhandledException, vous pouvez mettre à jour la classe pour souscrire à cet événement qui est appelé à chaque fois qu'une exception générée par l'application n'est pas gérée.

7. Le projet généré contient un document par défaut comprenant un code XAML définissant l'interface utilisateur (UI) de l'application. Pour visualiser ce fichier dans le designer, double-cliquez sur **MainPage.xaml** dans l'explorateur de solutions.

   Par défaut, le designer affiche le document de manière fractionnée. D'une part le code XAML et d'autre part une vue comprenant une représentation WYSIWYG des éléments de l'interface utilisateur. Hormis certains éléments intégrés au modèle pour afficher un nom et titre d'application, que vous pouvez supprimer si vous le souhaitez, le document XAML fournit une page blanche à laquelle vous êtes libre d'ajouter des contrôles pour créer votre propre interface utilisateur.

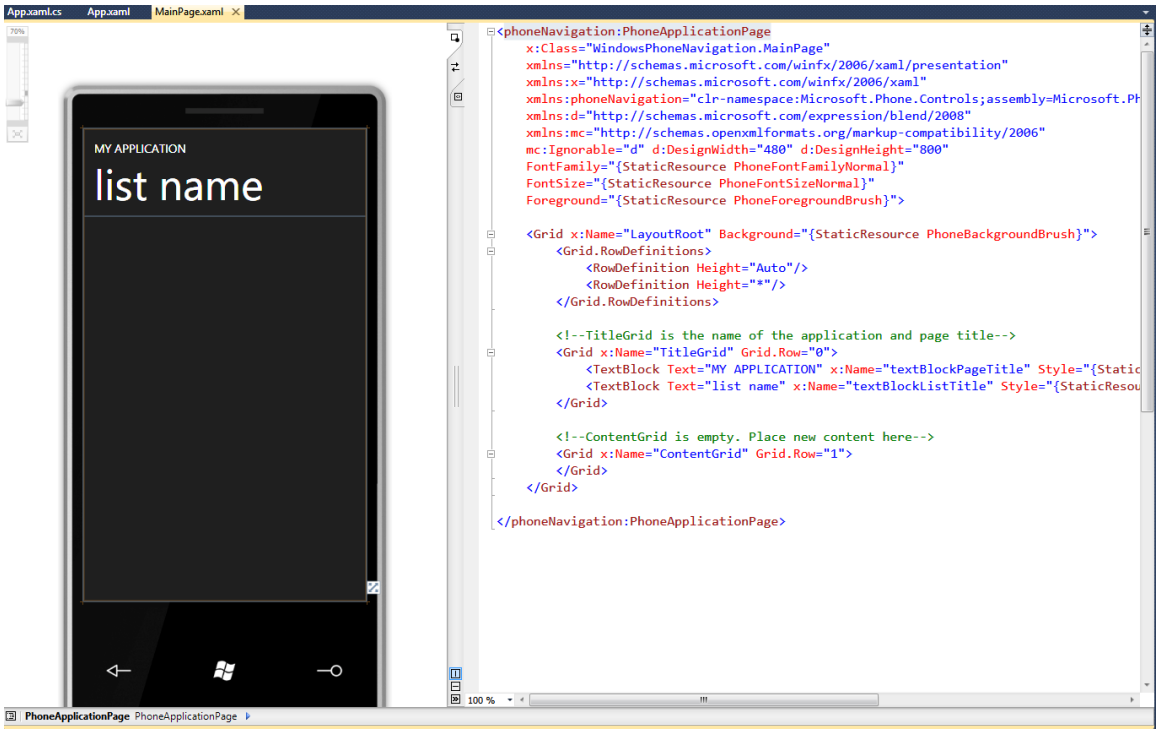

Figure 5

Le designer XAML présentant l'interface utilisateur principale de l'application

Note: Le XAML (Extensible Application Markup Language) est un langage déclaratif. Vous pouvez créer des éléments de l'interface utilisateur dans le code XAML, puis utiliser un fichier de code sous-jacent pour gérer des événements et manipuler les objets déclarés dans le XAML. Un langage déclaratif basé sur XAML est très intuitif

pour créer des interfaces, que ce soit dans le cadre de prototypes ou d'applications de production, notamment pour les personnes ayant des compétences en design et technologies Web.

8. Le fichier **ApplicationIcon.png** contient l'icône identifiant l'application dans l'écran de démarrage rapide du téléphone. Vous pouvez double-cliquer sur l'élément dans l'explorateur de solution pour ouvrir le fichier avec une application d'édition d'image tel que **Paint.exe**.

   Remarque : Dans Visual Studio 2010, double-cliquer sur le fichier icône dans l'explorateur de solution ouvre l'éditeur d'image.

9. Une application Windows Phone peut tirer parti des services fournis par la plateforme sous-jacente ou d'autres librairies. Pour cela, l'application doit référencer les *assemblies* correspondantes implémentant ces services. Pour afficher les *assemblies* référencées par le projet, dépliez les **References** dans l'explorateur de solution, et analysez la liste. Elle contient aussi bien des *assemblies* Silverlight classiques que des *assemblies* spécifiques à la plateforme Windows Phone.

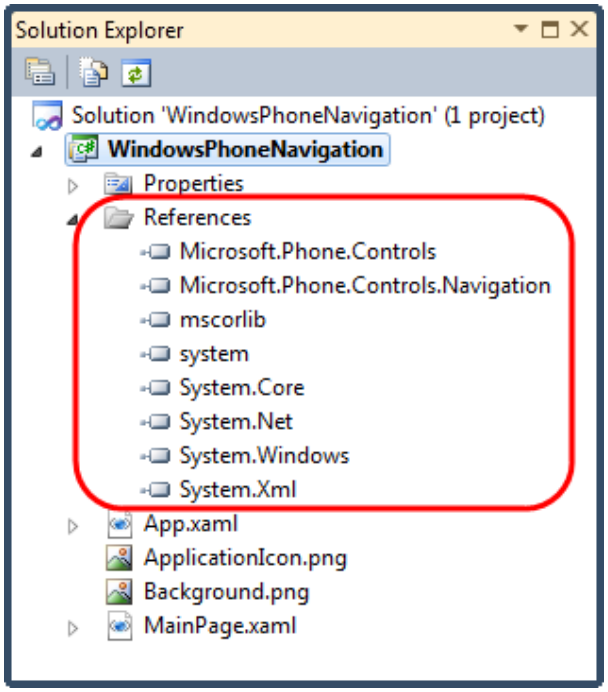

Figure 6

Les assemblies référencées dans le projet au sein de l'explorateur de solution

10. Ouvrez le fichier App.xaml (s'il ne l'est pas déjà) et ajoutez à la fin, juste avant la fin du tag Application.Resources, une chaîne de caractère partagée aux ressources de l'application que l'on peut référencer depuis toutes les pages XAML de l'application.

    (Extrait de code – *Navigation and Controls – Ex1 Tâche 1 Etape 10 – Application Name Resource*)

    XAML

```
<system:String x:Key="AppName">CONTROLS &amp; NAVIGATION</system:String>
```

11. Ouvrez MainPage.xaml et localisez la TitleGrid

    XAML

```
<Grid x:Name="TitleGrid" Grid.Row="0">
    <TextBlock Text="MY APPLICATION" x:Name="textBlockPageTitle"
Style="{StaticResource PhoneTextPageTitle1Style}"/>
    <TextBlock Text="list name" x:Name="textBlockListTitle"
Style="{StaticResource PhoneTextPageTitle2Style}"/>
</Grid>
```

12. Modifiez les valeurs de textBlockPageTitle et textBlockListTitle

    (Extrait de code – *Navigation and Controls –Ex1 Tâche 1 Etape 12 – Main Page TitleGrid*)

    XAML

```
<Grid x:Name="TitleGrid" Grid.Row="0">
  <TextBlock Text="{StaticResource AppName}" x:Name="textBlockPageTitle"
Style="{StaticResource PhoneTextPageTitle1Style}"/>
  <TextBlock Text="Main Page" x:Name="textBlockListTitle"
Style="{StaticResource PhoneTextPageTitle2Style}"/>
</Grid>
```

13. Appuyez sur **F5** pour lancer l'application dans l'émulateur Windows Phone.

    Notez qu'une fenêtre d'émulateur apparait, et qu'une pause a lieu alors que Visual Studio monte l'environnement de celui-ci et déploie l'image de l'application. Une fois prêt, l'émulateur présente la page de démarrage puis dans la foulée l'application.

Figure 2

Exécution de l'application dans l'émulateur Windows Phone

14. Jusqu'à la création de l'interface utilisateur et la programmation de la logique applicative, peu de choses peuvent être réalisées avec l'application à ce stade. Appuyez sur **SHIFT + F5** ou cliquez sur le bouton **Stop** de la barre d'outils pour sortir du mode debug. Ne fermez toutefois pas la fenêtre de l'émulateur.

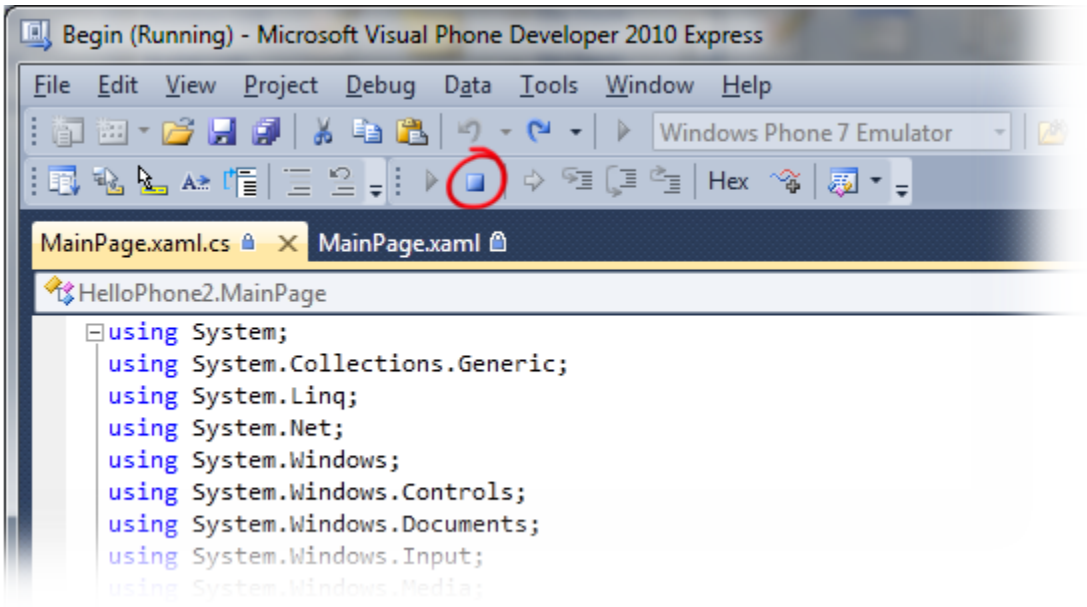

Figure 3

Terminer la session de débogage

> **Astuce :** Lorsque vous passez en mode debug, un délai est nécessaire pour monter l'environnement de l'émulateur et lancer l'exécution. Afin d'optimiser vos phases de débogage, **éviter de fermer l'émulateur** pendant que vous travaillez sur le code source dans Visual Studio. Une fois l'émulateur lancé, il est très rapide d'arrêter la session en cours, éditer le code source, puis compiler et déployer une nouvelle image de l'application pour refaire une passe de débogage.

15. Ajoutez 3 nouveaux dossiers au projet (Views, Misc, et Assets) pour gérer les éléments à afficher sur les pages du téléphone :

    a. Faîtes un clic droit sur le nom du projet

    b. Sélectionnez Add → New Folder

    c. Nommez **Views** le nouveau dossier

    d. Faîtes de même pour les dossiers **Misc** et **Assets**

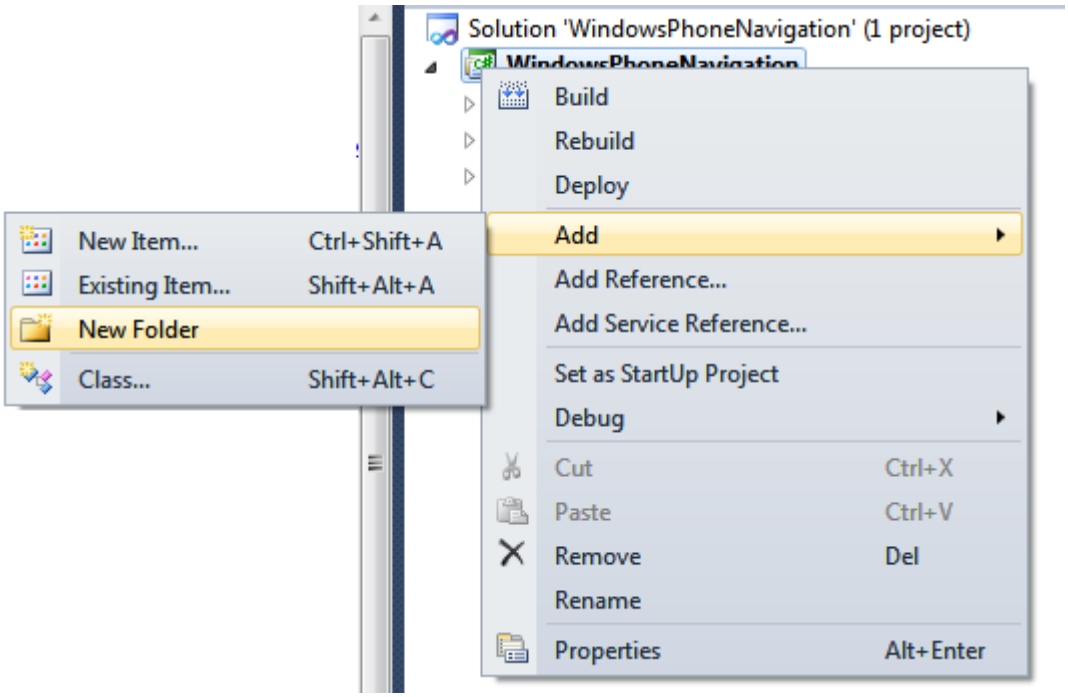

Figure 4
Ajout d'un nouveau dossier au projet

16. Faîtes un clic-droit sur le dossier **Views** et créez les sous-dossiers suivants :
    a. About
    b. Music
    c. Pictures
    d. Video
    e. WebBrowser

17. Faîtes un clic-droit sur le dossier **Assets** et créez un sous-dossier **Images**. Nommez le dossier **Views**.

> **Astuce** : Pour créer un sous-dossier, faites un clic-droit sur le dossier parent (**Views** dans le cas présent) et suivez les mêmes étapes que pour créer le dossier Views.

18. La structure du projet devrait désormais ressembler à la figure ci-dessous :

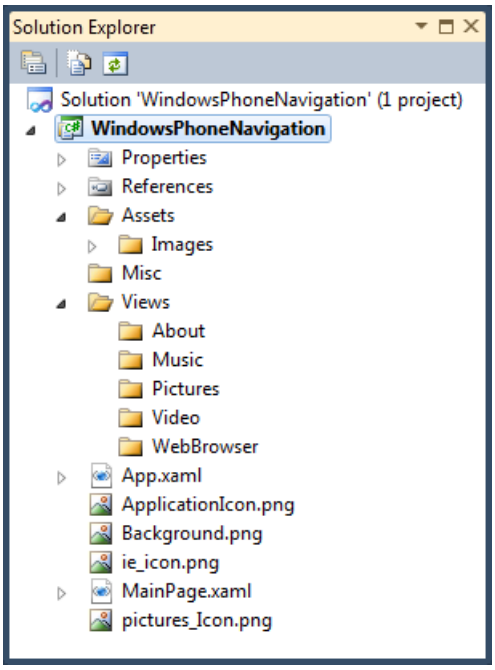

Figure 5

Structure du projet

Ajoutez des *assets* depuis le dossier correspondant. Le dossier Assets est situé dans le chemin d'installation de l'atelier (référencé [{LAB_PATH}\Assets])

Astuce : Pour ajouter un élément existant, faites un clic-droit sur le dossier désiré, et sélectionnez Add -> Existing Item. Dans la fenêtre "Add Existing Item" apparaissant, naviguez jusqu'à la localisation désirée, sélectionnez les éléments (potentiellement plusieurs) et cliquez sur le bouton "Add".

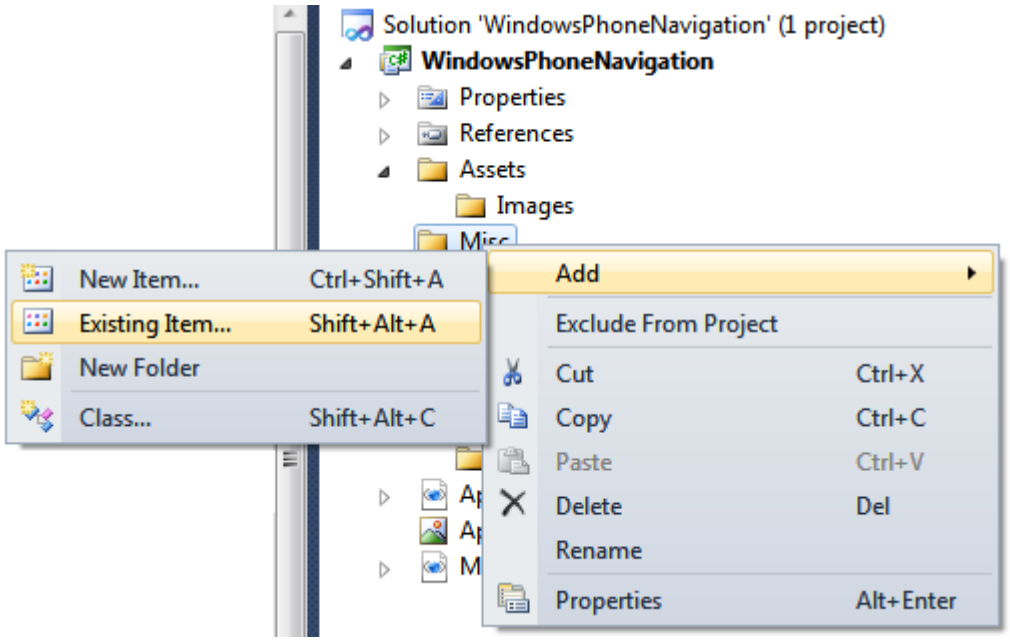

Figure 6

Ajout d'éléments existants aux dossiers du projet

Localisation de l'Asset	Nom de l'Asset	Dossier projet
{LAB_PATH}\Assets\Classes	Photo.cs Utils.cs	\Misc
{LAB_PATH}\Assets\Music	Music1.mp3 Music2.mid	\Assets
{LAB_PATH}\Assets\Video	Video1.wmv Video2.wmv	\Assets
{LAB_PATH}\Assets\Pictures	All the pictures	\Assets\Images
{LAB_PATH}\Assets\Icons	Ie_icon.png Pictures_Icon.png	\ (*root project folder*)

19. Silverlight supporte différents types de ressources : Embedded, Content, et Site of Origin. Les ressources Embedded sont compilées dans l'assembly; les ressources Content sont intégrées dans le package de l'application (fichier XAP), et les ressources site of Origin sont références depuis leur localisation d'origine. Dans cet atelier, toutes les ressources seront de type Content.
20. Passez tous les éléments ajoutées (à l'exception des fichiers.cs) en ressources Content :

- Faites un clic-droit sur le fichier ressource (par exemple, **Pictures_Icon.png**)
- Sélectionnez **Properties.**
- Sélectionnez **Content**.
- Faites de même pour tous les fichiers de ressources autres que les fichiers .cs

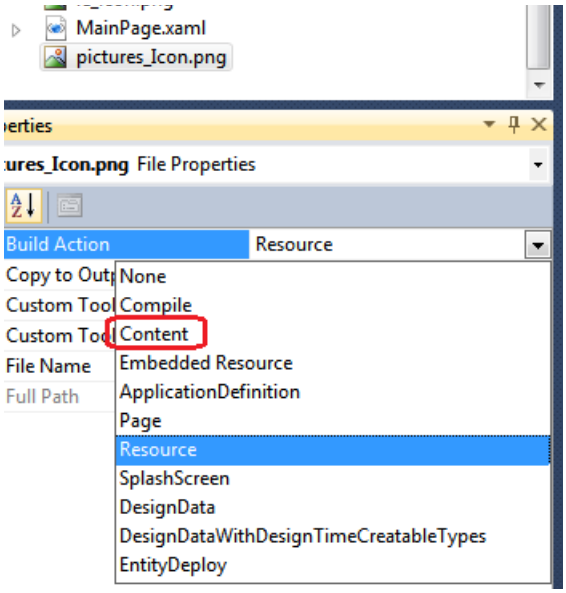

Figure 7

Modification du type de ressource

Après cette étape, la structure du projet devrait être la suivante :

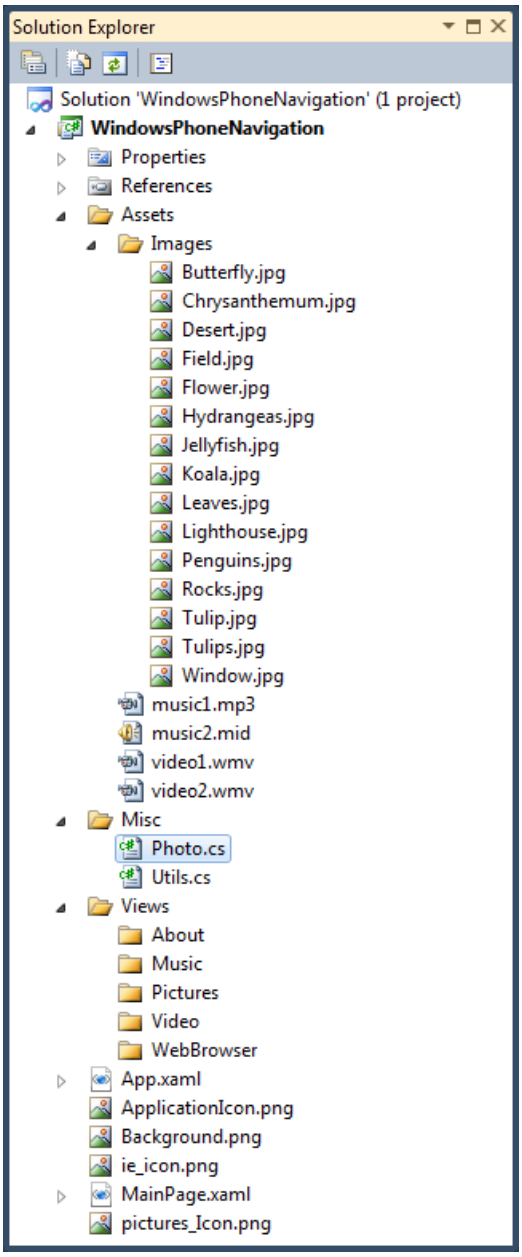

Figure 8

Structure du projet

> **Remarque** : Le nombre et les noms des images peuvent varier par rapport à la copie d'écran ci-dessus

Au cours de cette tâche, vous avez créé une nouvelle solution Windows Phone utilisant les modèles fournis dans Visual Studio 2010 for Windows Phone, et avez ajouté à l'application des

éléments fournis avec l'atelier. De plus, vous avez vu comment exécuter l'application dans l'émulateur Windows Phone.

Tâche 2 – Ajouter des Pages et naviguer entre elles

Dans cette tâche, vous ajouterez des pages et les modifierez. Ces pages seront utilisées ultérieurement pour lancer des fichiers musicaux et vidéo sélectionnés, affichez du contenu Web etc. Suite à cela, nous ajouterons un bouton **HyperlinkButton** à la page principale, MainPage, pour permettre la navigation.

Application.RootVisual (situé dans le fichier App.xaml) est une **PhoneApplicationFrame** qui contiendra une collection d'UriMappings que nous ajouterons dans cet exercice.

1. Sous le dossier Music, ajoutez un nouvel élément, **Windows Phone Portrait Page:**
   - Faites un clic-droit sur le dossier **Music**.
   - Sélectionnez **Add** -> **New Item**.
   - Sélectionnez **Windows Phone Portrait Page.**
   - Nommez l'élément **Default.xaml**.

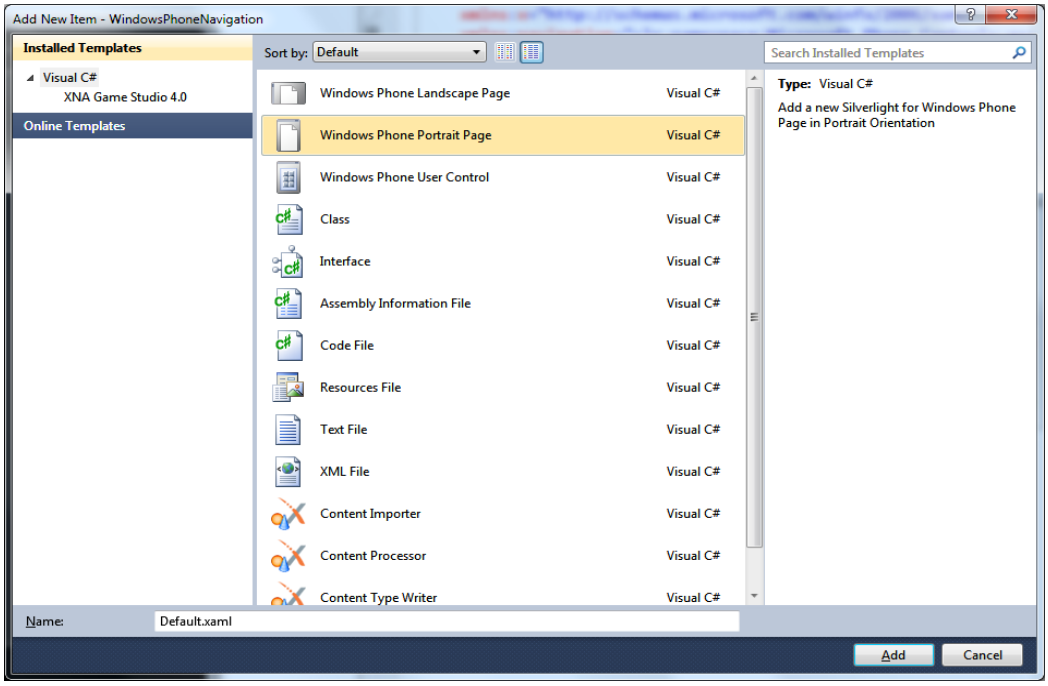

Figure 9

boite de dialogue Add New Item

2. Ouvrez la page nouvellement créée (si elle ne l'a pas été automatiquement)
3. Dans la page XAML, localisez la section "TitleGrid" et modifiez là ainsi

   (Extrait de code – *Navigation and Controls –Ex1 Tâche 2 Etape 3 – Music Page TitleGrid*)

XAML

```
<Grid Grid.Row="0" x:Name="TitleGrid">
    <TextBlock x:Name="ApplicationName" Text="{StaticResource AppName}"
Style="{StaticResource PhoneTextPageTitle1Style}"/>
    <TextBlock x:Name="ListName" Text="Music" Style="{StaticResource
PhoneTextPageTitle2Style}"/>
</Grid>
```

4. Suivez les étapes 1 à 3 pour créer une nouvelle page **Windows Phone Video Page** dans le dossier **Video**.

> **Remarque** : Le nom de la page name doit également être Default.xaml, mais le texte de celle-ci doit être "**Videos**" et non "*Music*"

5. Suivez les étapes 1 à 3 pour créer une nouvelle page **Windows Phone About Page** dans le dossier **About**.

> **Remarque** : Le nom de la page name doit également être Default.xaml, mais le texte de celle-ci doit être "**About**" et non "*Music*"

6. Ajoutez le markup suivant dans la "ContentGrid" de la page **About** :

   (Extrait de Code – *Navigation and Controls –Ex1 Tâche 2 Etape 6 – About Page Content Grid content*)

XAML

```
<TextBlock TextWrapping="Wrap" Margin="5" Style="{StaticResource
PhoneTextNormalStyle}" Text="This is a sample Windows Phone 7 navigation
application. All rights reserved to you - you, the writer of this
lab!"/>
```

7. Vous allez maintenant modifier MainPage.xaml en y ajoutant des contrôles **HyperlinkButton** pour naviguer entre les pages. Ouvrez App.xaml et localisez la ressource AppName créée précédemment. Ajoutez le code suivant :

(Extrait de Code – *Navigation and Controls – Ex1 Tâche 2 Etape 7 – Hyperlink Style*)

XAML

```
<!-- Hyperlink style -->
<Style x:Key="PhoneHyperlinkStyle" TargetType="HyperlinkButton">
  <Setter Property="FontFamily" Value="{StaticResource
PhoneFontFamilyNormal}"/>
  <Setter Property="FontSize" Value="{StaticResource
PhoneFontSizeSmall}"/>
  <Setter Property="Margin" Value="30,10,0,10"/>
</Style>
```

8. Intégrez un nouveau *namespace* à l'application en ajoutant la ligne suivante dans la balise Application (par exemple après l'attribut **phoneNavigation**, avant la fermeture de la balise).

XAML

```
xmlns:nav2="clr-
namespace:System.Windows.Navigation;assembly=Microsoft.Phone.Controls.Na
vigation"
```

> **Remarque**: La déclaration *clr-namespace* est requise pour intégrer le *namespace* définit dans l'assembly .NET.
>
> La syntaxe prend en compte les valeurs suivantes :
>
> **clr-namespace:** Le namespace du *common language runtime* (CLR) declaré dans l'assembly contenant les types publics.
>
> **assembly=** L'assembly contenant tout ou partie des namespace du CLR. Classiquement, cette valeur est simplement le nom de l'assembly, non son chemin, sans l'extension (.dll ou .exe). Le chemin de cette assembly doit être défini comme une référence du projet dans le fichier contenant le XAML que l'on souhaite lier. Afin d'intégrer du versioning et une signature, la valeur d'assembly peut être une chaine de caractère telle que définie par **AssemblyName**, plutôt qu'un simple nom de chaine de caractère.

9. Ouvrez le fichier App.xaml, localisez la PhoneApplicationFrame nommée "**RootFrame**" et ajoutez l'objet **UriMapper**. L'objet PhoneApplicationFrame définit un élément

UriMapper qui sera utilisé dans toute l'application pour la navigation, et qui devrait être comme ci-dessous.

(Extrait de Code – *Navigation and Controls – Ex1 Tâche 2 Etape 9 – UriMapper element*)

XAML

```
<phoneNavigation:PhoneApplicationFrame x:Name="RootFrame"
Source="/MainPage.xaml">
  <phoneNavigation:PhoneApplicationFrame.UriMapper>
    <nav2:UriMapper>
      <nav2:UriMapper.UriMappings>
        <nav2:UriMapping Uri="/Music/{song}"
MappedUri="/Views/Music/Default.xaml?Song=Assets/{song}"/>
        <nav2:UriMapping Uri="/Video/{video}"
MappedUri="/Views/Video/Default.xaml?Video=Assets/{video}"/>
        <nav2:UriMapping Uri="/WebBrowser/{defaultURL}"
MappedUri="/Views/WebBrowser/Default.xaml?DefaultURL={defaultURL}"/>
        <nav2:UriMapping Uri="/Pictures"
MappedUri="/Views/Pictures/Default.xaml"/>
        <nav2:UriMapping Uri="/PictureView/{picture}"
MappedUri="/Views/Pictures/PictureView.xaml?Picture={picture}"/>
        <nav2:UriMapping Uri="/About"
MappedUri="/Views/About/Default.xaml"/>
      </nav2:UriMapper.UriMappings>
    </nav2:UriMapper>
  </phoneNavigation:PhoneApplicationFrame.UriMapper>
</phoneNavigation:PhoneApplicationFrame>
```

10. Ouvrez MainPage.xaml, localisez "ContentGrid" et ajoutez-y le code suivant :

    (Extarit de Code – *Navigation and Controls –Ex1 Tâche 2 Etape 10 – MainPage ContentGrid content*)

XAML

```
<ScrollViewer BorderBrush="Transparent">
  <StackPanel Margin="5">
    <TextBlock Text="Music" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
    <HyperlinkButton NavigateUri="/Music/music1.mp3" Content="Song #1"
Style="{StaticResource PhoneHyperlinkStyle}" />
    <HyperlinkButton NavigateUri="/Music/music2.mid" Content="Song #2"
Style="{StaticResource PhoneHyperlinkStyle}" />

    <TextBlock Text="Videos" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
```

```
    <HyperlinkButton NavigateUri="/Video/video1.wmv" Content="Video #1"
Style="{StaticResource PhoneHyperlinkStyle}"/>
    <HyperlinkButton NavigateUri="/Video/video2.wmv" Content="Video #2"
Style="{StaticResource PhoneHyperlinkStyle}"/>

    <TextBlock Text="About" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
    <HyperlinkButton NavigateUri="/About" Content="About"
Style="{StaticResource PhoneHyperlinkStyle}"/>
  </StackPanel>
</ScrollViewer>
```

A ce stade, l'apparence de la surface Visual Studio devrait être la suivante :

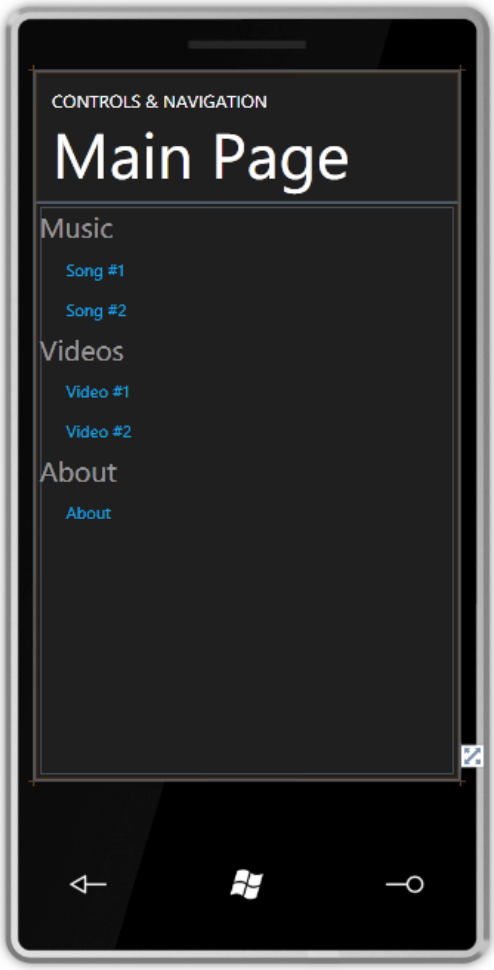

Figure 10

Surface de design

11. Compilez et lancez l'application. Cliquez sur les hyperliens pour vous assurer que tout fonctionne correctement.

    Remarque : Utiliser le bouton Back pour retourner à la page principale de l'application.

    Cette étape conclue le 1er exercice. Durant celui-ci, vous avez créé une application basique Silverlight pour Windows Phone, y avez ajouté de nombreuses ressources et pages, et avez créé la navigation entre ces pages. Dans le prochain exercice, vous ajouterez des contrôles utilisateur aux pages créées.

    Remarque : La solution de cet exercice est situé au chemin suivant: *{LAB_PATH}\Ex1-IntroductionToPhoneNavigationModel\End*

Exercice 2: Introduction aux contrôles disponibles pour les applications Windows Phone

Dans cette section, vous allez ajouter des fonctionnalités média à vos pages Musique et Vidéo déjà créées, ajouter de nouvelles pages aux images existantes, et utiliser les contrôles WebBrowser.

Remarque : Vous pouvez continuer à travailler sur la solution créée précédemment. Sinon, si vous n'avez pas réalisé l'exercice précédent, vous pouvez ouvrir une solution existante à l'emplacement suivant : *{LAB_PATH}\Ex2-IntroductionToTheControls\Begin*

Tâche 1 – Ajouter des éléments media aux fichiers Play Media Files

Au cours de cette tâche, vous ajouterez 2 nouvelles pages au projet. Ces pages seront utilisées afin de présenter du contenu vidéo et jouer des fichiers musicaux fournis avec l'atelier.

1. Ouvrez la solution créée dans l'exercice précédent
2. Ouvrez le fichier **Default.xaml** depuis le dossier Vidéo
3. Ajoutez l'extrait de code suivant à "ContentGrid":

(Extrait de Code – *Navigation and Controls – Ex2 Tâche 1 Etape 3 – Video Page ContentGrid content*)

XAML

```
<MediaElement x:Name="media" HorizontalAlignment="Stretch"
AutoPlay="True" VerticalAlignment="Stretch" Stretch="Uniform" />
```

Les propriétés d'alignement horizontal et vertical correspondent aux caractéristiques d'alignement d'un FrameworkElement lorsqu'il est constitué au sein d'un parent d'affichage, tel qu'un contrôle de type *panel* ou *items control*. Les valeurs correspondantes sont Left, Center, Right, Stretch for HorizontalAlignment, Top, Center, Bottom, and Stretch for VerticalAlignment. La propriété Stretch étire l'élément de sorte qu'il utilise tout l'espace d'affichage disponible de l'élément parent.

La propriété Stretch est très importante lorsque l'on utilise des éléments media comme des Images. Elle détermine comment le contenu est redimensionné afin de remplir son espace alloué. Les options possible pour cette propriété sont :

- **None** – Le contenu conserve sa taille initiale
- **Fill** - Le contenu est redimensionné afin d'utiliser tout l'espace disponible sans conservation des proportions d'affichage initiales
- **Uniform** - Le contenu est redimensionné afin d'utiliser tout l'espace disponible tout en conservant ses proportions d'affichage initiales
- **UniformToFill** – Le contenu d'origine est redimensionné afin d'utiliser tout l'espace disponible tout en conservant ses proportions d'affichage initiales et, si nécessaire, est rogné pour tenir dans les dimensions cibles.

Remarque : Vous pouvez également utiliser la surface de design interactive pour ajouter le MediaElement. Pour cela, ouvre les boite à outils des contrôles, sélectionnez le MediaElement dans la liste et faites un glisser-déposer sur la surface de design.

4. Ouvrez le fichier **Default.xaml**

Remarque : Pour ouvrir le fichier de code sous-jacent, faites un clic-droit sur le fichier XAML dans l'explorateur de solution et sélectionnez Video Code.

5. Vous devez maintenant mettre à jour la source de l'élément media ajouté selon les paramètres reçus des services de navigation. Pour cela, souscrivez tout d'abord à

l'événement Loaded de la page. Ajoutez le code suivant après la fonction *InitializeComponent* :

C#

```
this.Loaded += new RoutedEventHandler(Default_Loaded);
```

6. Ajoutez la fonction de gestionnaire d'événement :

C#

```
void Default_Loaded(object sender, RoutedEventArgs e)
{
}
```

> **Astuce** : Vous pourriez faire afficher par Visual Studio 2010 le gestionnaire d'événement par défaut en appuyant 2 fois sur la touche Tab après *+=* dans la souscription à l'événement.

7. Ajoutez le code suivant à la fonction de gestionnaire d'événement :

(Extrait de Code – *Navigation and Controls – Ex2 Tâche 1 Etape 7 – Video Default_Loaded event handler code*)

C#

```
if (NavigationContext.QueryString.Count > 0)
{
    try
    {
        media.Source = new
Uri(NavigationContext.QueryString.Values.First(),
UriKind.RelativeOrAbsolute);
        media.Position = TimeSpan.FromMilliseconds(0);
        media.Play();
    }
    catch (Exception ex)
    {
        //handle the exception;
    }
}
```

8. Modifiez le fichier Default.xaml dans le dossier Music. Localisez “ContentGrid” et ajoutez le code suivant :

   (Extrait de Code – *Navigation and Controls –Music Page ContentGrid content*)

XAML

```
<MediaElement x:Name="media"/>
<TextBlock x:Name="musicToPlay" Style="{StaticResource
PhoneTextNormalStyle}"/>
```

9. Ouvrez le fichier de code sous-jacent afin de le modifier.

10. Souscrivez à l’événement Loaded de la page. Ajoutez le code suivant après la fonction *InitializeComponent* :

#

```
this.Loaded += new RoutedEventHandler(Default_Loaded);
```

11. Ajoutez la fonction :

C#

```
void Default_Loaded(object sender, RoutedEventArgs e)
{
}
```

12. Ajoutez le code suivant à la fonction de gestionnaire d’événement :

    (Extrait de Code– *Navigation and Controls – Ex2 Tâche 1 Etape 12 – Music Default_Loaded event handler code*)

C#

```
if (NavigationContext.QueryString.Count > 0)
{
    this.musicToPlay.Text =
this.NavigationContext.QueryString.Values.First();

    try
    {
        media.AutoPlay = true;
        media.Source = new
Uri(NavigationContext.QueryString.Values.First(),
UriKind.RelativeOrAbsolute);
        media.Position = TimeSpan.FromMilliseconds(0);
```

```
            media.Play();
        }
        catch (Exception ex)
        {
            //handle the exception;
        }
}
```

13. Compilez et lancez l'application. Naviguez jusqu'aux pages Vidéo et/ou Music pour vous assurer que cela fonctionne.

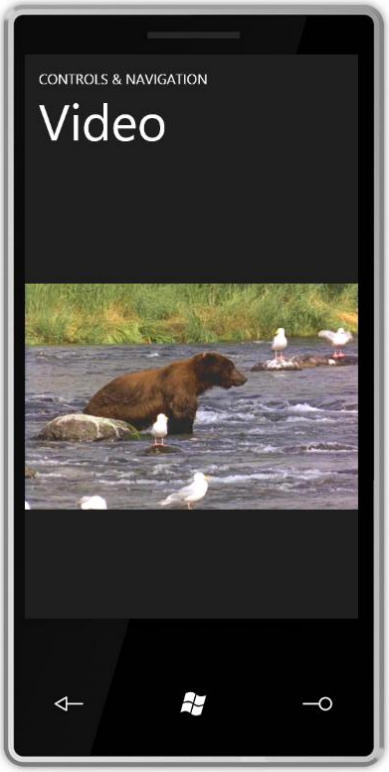

Figure 11

Page video

14. Arrêtez le débogage et retourner au code de l'application.

Au cours de cette tâche, vous avez ajouté 2 pages à l'application, géré l'événement Loaded de la page, et créé la logique applicative basée sur les paramètres de requêtes fournis.

Tâche 2 – Utiliser des ListBox pour afficher des Images

Durant cette tâche, vous créerez une nouvelle page affichant des images intégrées dans le package XAP. Les images seront affichées dans un contrôle ListBox. Plus loin dans cet atelier, vous créerez une page PictureView pour afficher les images unitaires.

Remarque : La version actuelle ne prend pas en charge l'acquisition de photos depuis la librairie de photos du téléphone ou depuis l'appareil directement. Cela sera le cas ultérieurement.

Dans cette tâche, vous ajouterez également une nouvelle page contenant un contrôle ListBox permettant d'afficher des photos.

1. Ajoutez une nouvelle page Default.xaml dans le dossier **Pictures**.
2. Modifiez le "TitleGrid" comme présenté dans le code ci-dessous :

   (Extrait de Code – *Navigation and Controls – Ex2 Tâche 2 Etape 2 – Pictures Page TitleGrid content*)

   XAML

```
<TextBlock Text="{StaticResource AppName}" Style="{StaticResource
PhoneTextPageTitle1Style}"/>
<TextBlock Text="Images" Style="{StaticResource
PhoneTextPageTitle2Style}"/>
```

3. Modifiez le "ContentGrid" comme présenté dans le code ci-dessous :

   (Extrait de Code – *Navigation and Controls – Ex2 Tâche 2 Etape 3 – Pictures Page ContentGrid content*)

   XAML

```
<StackPanel Orientation="Vertical">
  <StackPanel Orientation="Horizontal" Margin="5">
    <Button x:Name="btnRemoveSelection" Content="Remove Image"
Click="btnRemoveSelection_Click" IsEnabled="False"/>
  </StackPanel>

  <ListBox x:Name="lstPictures" Width="450" Height="520" Margin="10"
SelectionChanged="lstPictures_SelectionChanged">
    <ListBox.ItemTemplate>
      <DataTemplate>
        <StackPanel Orientation="Horizontal">
          <Image Source="{Binding Image}" Width="100" Stretch="Uniform"
HorizontalAlignment="Center"/>
```

```
            <TextBlock Text="{Binding Filename}" TextWrapping="Wrap" />
          </StackPanel>
        </DataTemplate>
      </ListBox.ItemTemplate>
    </ListBox>
</StackPanel>
```

Cet extrait de code ajoute des contrôles Button et ListBox. La ListBox définit le DataTemplate pour les éléments de ListBox. Le DataTemplate utilise le mécanisme de DataBinding pour traiter les données à l'exécution.

4. A ce stade, le design de la page devrait ressembler à cela :

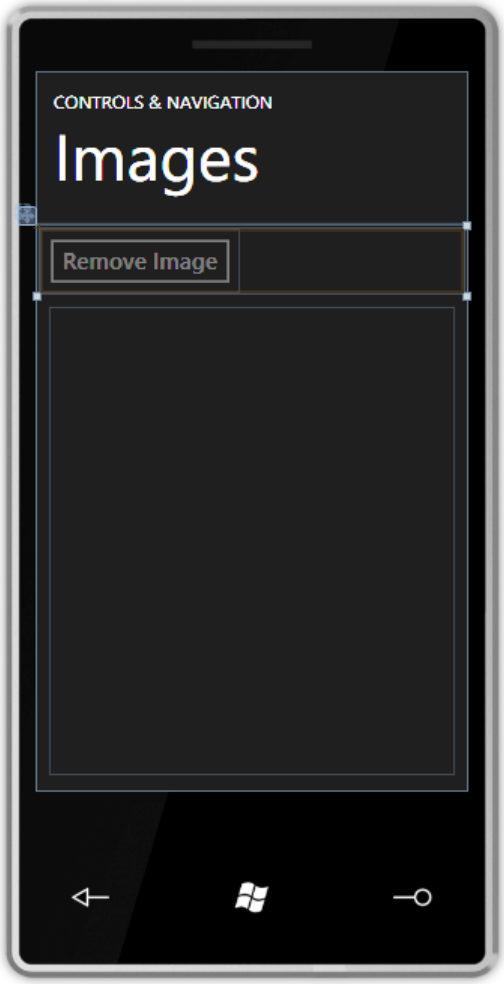

Figure 12

Design de la page Images

5. Ouvrez le fichier de code sous-jacent.
6. Ajoutez les instructions suivantes :

C#

```
using System.Collections.ObjectModel;
using WindowsPhoneNavigation.Misc;
```

7. Afin d'utiliser le mécanisme de data binding avec les collections et recevoir des notifications quand elles changent, Silverlight utilise des événements. CollectionChanged est l'événement attendu par le moteur de data binding lorsqu'une collection change. Il est défini ainsi :

C#

```
event NotifyCollectionChangedEventHandler CollectionChanged
```

L'implémentation de la collection doit déclencher cet événement à chaque changement de celle-ci (Ajout/Edition/Suppression d'éléments, réorganisation etc.).

Cet événement est défini dans l'interface INotifyCollectionChanged. Pour des mises à jours automatiques du data binding sur la collection, vous devez créer votre propre collection et implémenter cette interface. Vous pouvez également utiliser des collections fournies dans la librairie de classe Silverlight qui implémentent cette interface. Silverlight fournit 2 classes/collections qui implémentent cette interface: ObservableCollection<T> et ReadOnlyObservableCollection<T>.

ObservableCollection – Représente une collection de données dynamique fournissant des notifications lorsque des éléments sont ajoutés, supprimés, ou lorsque la liste entière est rafraichie.

ReadOnlyObservableCollection - Représente une ObservableCollection en lecture seule.

Dans les 2 cas, le mécanisme de data binding répondra aux événements levés en mettant à jour les objets liés à la collection.

8. Ajoutez ObservableCollection pour lier les images à la classe.

C#

```
ObservableCollection<Photo> photos = new ObservableCollection<Photo>();
```

9. Ajoutez une fonction pour initialiser la collection de photos.

(Extrait de Code – *Navigation and Controls –Ex 2 Tâche 2 Etape 9 – Photo Collection initialization method*)

C#

```
private void InitializePhotos()
{
    photos.Add(new Photo()
    {
        Filename = "Butterfly.jpg",
        Image = Utils.GetImage("Butterfly.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Chrysanthemum.jpg",
        Image = Utils.GetImage("Chrysanthemum.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Desert.jpg",
        Image = Utils.GetImage("Desert.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Field.jpg",
        Image = Utils.GetImage("Field.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Flower.jpg",
        Image = Utils.GetImage("Flower.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Hydrangeas.jpg",
        Image = Utils.GetImage("Hydrangeas.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Jellyfish.jpg",
        Image = Utils.GetImage("Jellyfish.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Koala.jpg",
        Image = Utils.GetImage("Koala.jpg")
    });
```

```
    photos.Add(new Photo()
    {
        Filename = "Leaves.jpg",
        Image = Utils.GetImage("Leaves.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Lighthouse.jpg",
        Image = Utils.GetImage("Lighthouse.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Penguins.jpg",
        Image = Utils.GetImage("Penguins.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Rocks.jpg",
        Image = Utils.GetImage("Rocks.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Tulip.jpg",
        Image = Utils.GetImage("Tulip.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Tulips.jpg",
        Image = Utils.GetImage("Tulips.jpg")
    });
    photos.Add(new Photo()
    {
        Filename = "Window.jpg",
        Image = Utils.GetImage("Window.jpg")
    });
}
```

10. Appelez la fonction depuis la page et définissez la collection initialisée comme ItemsSource d'une ListBox :

C#

```
InitializePhotos();
lstPictures.ItemsSource = photos;
```

11. Ajoutez une fonction de gestionnaire d'événement pour gérer l'événement Click sur le bouton **Remove** :

    (Extrait de Code – *Navigation and Controls –Ex2 Tâche 2 Etape 11 – Remove Button Click event handler*)

C#

```
private void btnRemoveSelection_Click(object sender, RoutedEventArgs e)
{
    if (null != lstPictures.SelectedItem)
        photos.Remove(lstPictures.SelectedItem as Photo);
}
```

12. Ajoutez une fonction de gestionnaire d'événement pour gérer l'événement SelectionChanged de la **ListBox**

    (Extrait de Code – *Navigation and Controls –Ex2 Tâche 2 Etape 12 – Pictures Listbox SelectionChanged event handler*)

C#

```
private void lstPictures_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (null != lstPictures.SelectedItem)
    {
        btnRemoveSelection.IsEnabled = true;
    }

    if (photos.Count == 0)
        btnRemoveSelection.IsEnabled = false;
}
```

13. Ouvrez App.xaml et ajoutez l'objet suivant qui pointe vers la localisation de l'image, après l'objet **AppName** :

XAML

```
<system:String x:Key="ImagesLocation">Assets/Images/</system:String>
```

14. Ajoutez dans MainPage.xaml un **Hyperlink** à une nouvelle page (après Vidéos, avant la page About) :

(Extrait de Code – *Navigation and Controls – Ex2 Tâche 2 Etape 14 – MainPage pictures Hyperlink*)

XAML

```
<TextBlock Text="Images" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
<HyperlinkButton NavigateUri="/Pictures" Content="Images"
Style="{StaticResource PhoneHyperlinkStyle}"/>
```

15. Compilez et lancez l'application. Naviguer jusqu'à la page des Images et validez son fonctionnement.

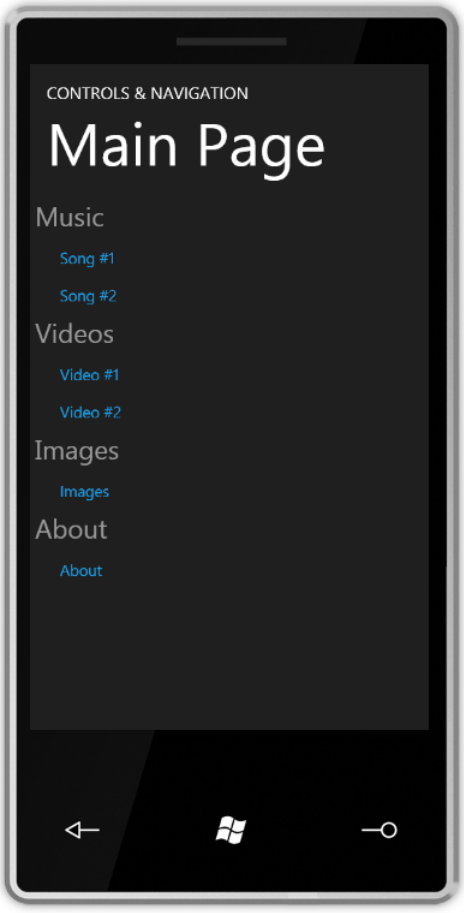

Figure 13

L'application après finalisation de la tâche 2

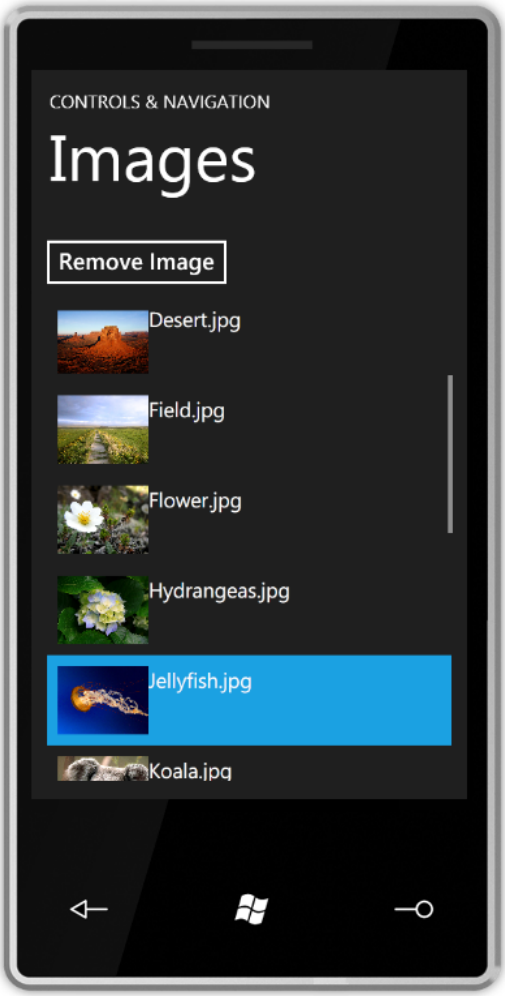

Figure 14

Page Images

16. Arrêtez le débogage et retournez au code de l'application.
    Au cours de cette étape, vous avez ajouté une nouvelle page à l'application, au sein de laquelle vous avez créé une ListBox pour gérer les photos fournies avec l'atelier. La ListBox était liée à l'ObservableCollection afin de permettre des mises à jour tout lors des ajouts/suppressions d'images à la collection. De plus, vous avez créé un DataTemplate pour afficher chaque élément dans une ListBox avec un titre à côté de l'icône de l'image.

Tâche 3 – Concevoir une Navigateur Web Simple

Dans cette tâche, vous ajouterez une nouvelle page avec un WebBrowser et d'autres contrôles additionnels. Tous ensemble, ces contrôles constitueront une interface de navigation web simple.

Dans cette tâche, vous ajouterez également une nouvelle page de type WebBrowser qui sera utilisée pour créer un navigateur web simple.

1. Ajoutez une nouvelle page Landscape au dossier **WebBrowser** et nommez le Default.xaml.
2. Ajoutez une référence à l'assembly Microsoft.Phone.Controls.WebBrowser :

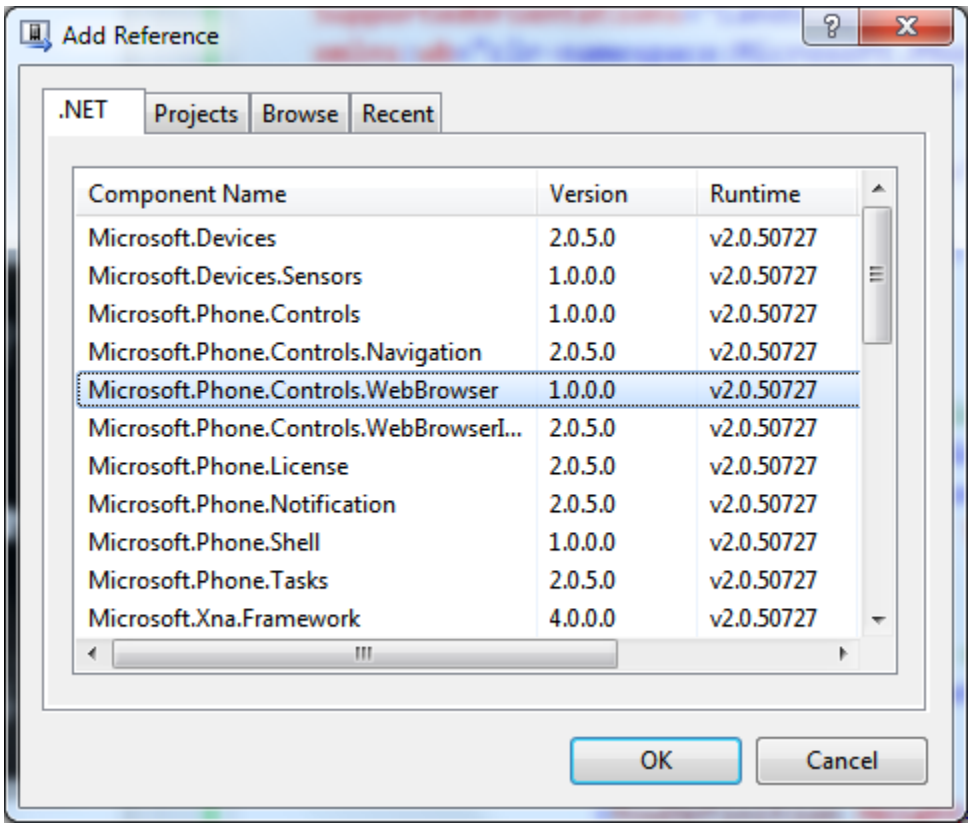

Figure 15

Ajout d'une référence à l'assembly Microsoft.Phone.Controls.WebBrowser

3. Ajoutez un nouveau namespace avec le contrôle WebBrowser. Ajoutez l'extrait suivant à l'objet PhonePageApplication (avant l'attribut SupportedPageOrientation mais avant la balise de fermeture de l'objet)

XAML

```
xmlns:wb="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Web
Browser"
```

4. Modifiez “TitleGrid” comme dans l’extrait ci-dessous:

   (Extrait de Code – *Navigation and Controls – Ex2 Tâche 3 Etape 4 – WebBrowser page TitleGrid content*)

XAML

```
<TextBlock x:Name="ApplicationName" Text="{StaticResource AppName}"
Style="{StaticResource PhoneTextPageTitle1Style}"/>
<TextBlock x:Name="ListName" Text="Web Browser" Style="{StaticResource
PhoneTextPageTitle2Style}"/>
```

5. Ajoutez l’extrait suivant à “ContentGrid”:

   (Extrait de Code – *Navigation and Controls – Ex2 Tâche 3 Etape 5 – WebBrowser page ContentGrid content*)

XAML

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="80"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <StackPanel Orientation="Horizontal">
    <TextBox x:Name="txtURL" Margin="5" Width="400"
Style="{StaticResource PhoneTextBoxStyle}"/>
    <Button Content="Go" Click="Button_Click"/>
  </StackPanel>
  <wb:WebBrowser Grid.Row="1" x:Name="wb" Margin="10"/>
</Grid>
```

6. L’apparence devrait désormais être la suivante :

Figure 16

Design de la page WebBrowser

7. Ouvrez le fichier de code sous-jacent de la page
8. Ajoutez la constant httpPrefix à la classe

```
C#
const string httpPrefix = "http://";
```

9. Souscrivez à l'évènement Loaded pour gérer les paramètres de navigation. Ajoutez le code suivant à la classe :

```
C#
this.Loaded += new RoutedEventHandler(Default_Loaded);
```

10. Créez la fonction de gestion de l'évènement appropriée et ajoutez la logique métier. Le code devrait in fine s'apparenter à :

    (Extrait de Code – *Navigation and Controls – Ex2 Tâche 3 Etape 10 – WebBrowser Loaded event handler*)

```
C#
void Default_Loaded(object sender, RoutedEventArgs e)
{
    if (NavigationContext.QueryString.Count > 0)
    {
```

```
        string urlToNavigate =
NavigationContext.QueryString.Values.First();

        if (urlToNavigate.IndexOf(httpPrefix) == -1)
            urlToNavigate = httpPrefix + urlToNavigate;

        Uri uri = new Uri(urlToNavigate, UriKind.Absolute);

        if (null != uri)
            wb.Navigate(uri);
    }
}
```

Le morceau de code vérifie si le paramètre de navigation existe et fournit une adresse URL valide (selon le format d'URL *http://www.somesite.com/somefolder/somepage.html*). Si tel est le cas, le téléphone accède à l'URL spécifiée.

11. Ajoutez la fonction de gestion de l'évènement click du bouton « Go ». La fonctionnalité est similaire à la précédente. Récupérez l'URL désirée depuis la **TextBox** sur l'écran et après validation, allez à sa localisation :

    (Extrait de Code – *Navigation and Controls – Ex2 Tâche 3 Etape 11 – Gestion d'évènemen tclick du bouton Go*)

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    string urlToNavigate =
       txtURL.Text.IndexOf(httpPrefix) > -1 ?
       txtURL.Text : httpPrefix + txtURL.Text;
    Uri uri = new Uri(urlToNavigate, UriKind.Absolute);

    if (null != uri)
        wb.Navigate(uri);
}
```

12. Dans MainPage.xaml, ajoutez **Hyperlink** à une nouvelle page (après Image, avant la page About) :

(Extrait de Code – *Navigation and Controls – Ex2 Tâche 3 Etape 12 – MainPage WebBrowser Hyperlink*)

XAML

```
  <TextBlock Text="Web" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
  <HyperlinkButton NavigateUri="/WebBrowser/www.bing.com" Content="Web
Browser" Style="{StaticResource PhoneHyperlinkStyle}"/>
```

13. Compilez et lancez l'application.

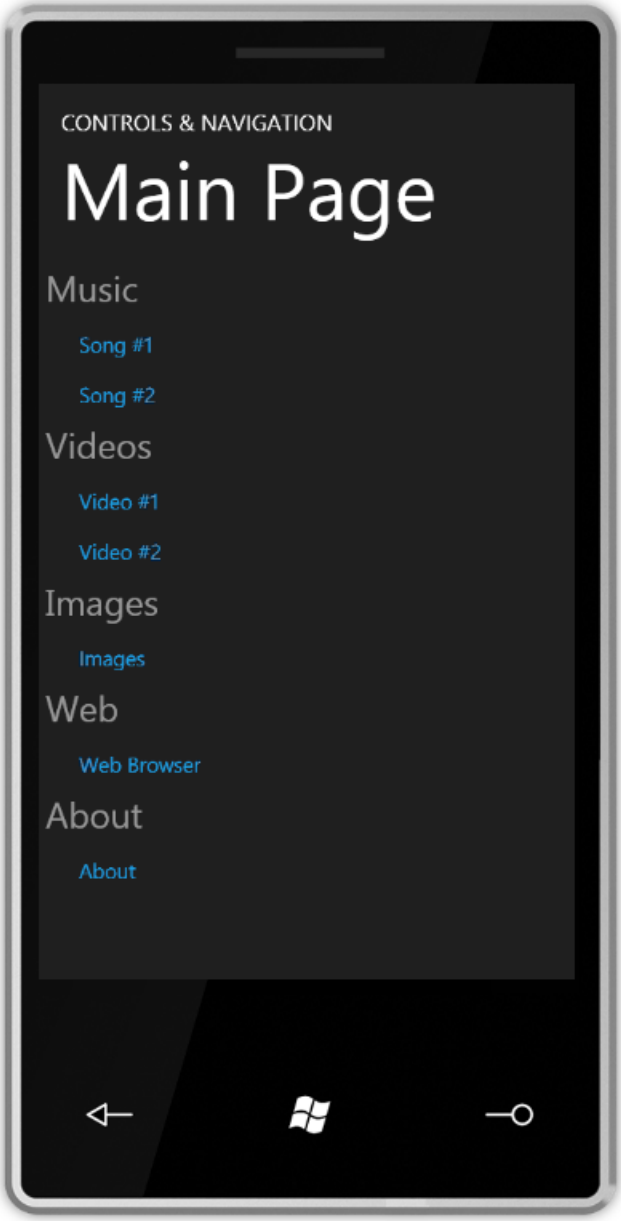

Figure 17

Page principale une fois cette tâche terminée

14. Allez sur la page du navigateur web et vérifiez son fonctionnement.

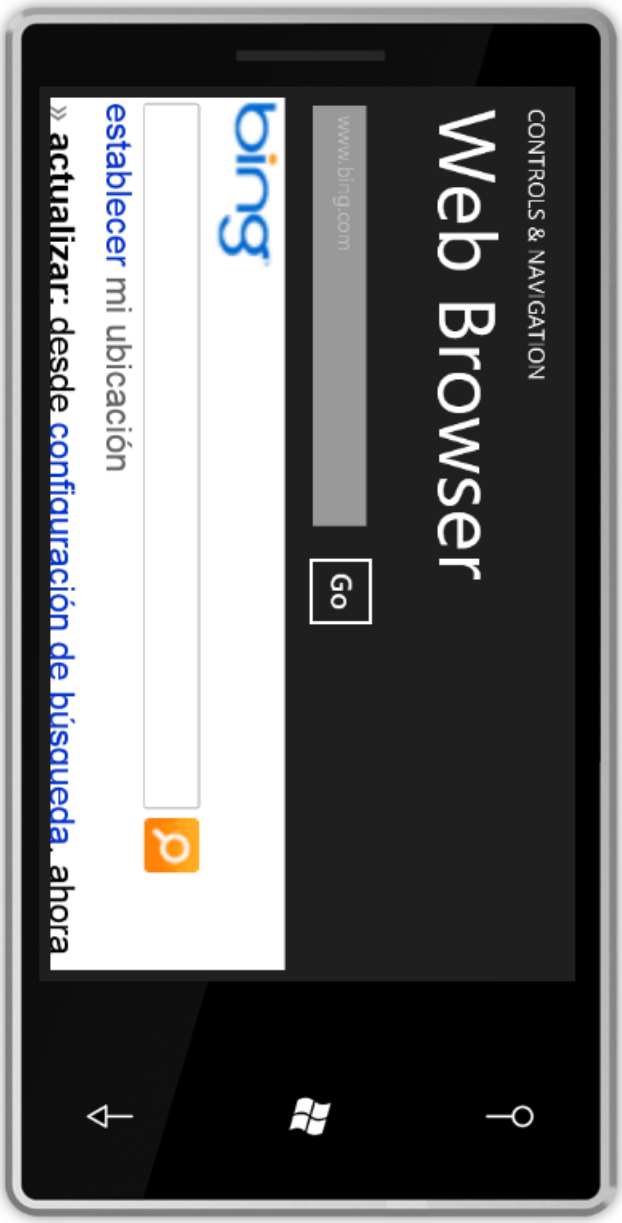

Figure 18
Web BrowserPage within wrongly oriented emulator window

15. Lorsque vous allez sur la page du navigateur web, l'émulateur l'ouvre au format paysage, même si l'émulateur est lui en portrait. Pour modifier l'orientation, cliquez sur les boutons « Rotate » :

Figure 19

Changer l'orientation de l'émulateur

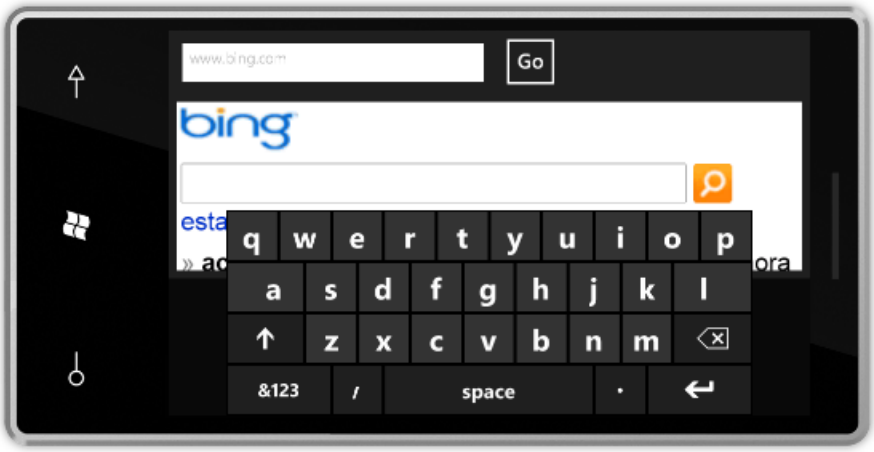

Figure 20

Navigateur web au format paysage

Cette étape achève le second exercice.

Durant cet exercice, vous avez ajouté une fonctionnalité de playback à vos pages Music et Video existantes, utilisé des contrôles pour ajouter de nouvelles pages aux photos, et utilisé les contrôles WebBrowser.

Remarque : La solution de cet exercice est situé : {LAB_PATH}\Ex2-IntroductionToTheControls\End

Exercice 3 : Introduction aux services Windows Phone

Dans cet exercice, vous découvrirez des fonctionnalités spécifiques au téléphone, notamment comment :

- Définir l'orientation des pages
- Gérer les évènements de modification de l'orientation
- Gérer l'évènement de pression du bouton « Back »
- Ajouter une Application Bar à votre application

Remarque : Vous pouvez continuer à vous baser sur la solution créée dans l'exercice précédent. Sinon, si vous n'avez pas fait cet exercice, vous pouvez ouvrir la solution située au chemin suivant : *{LAB_PATH}\Ex3-IntroductionToWindowsPhoneServices\Begin*

Tâche 1 – Gérer les modifications d'orientation des pages

Au cours de cette tâche, vous apprendrez à gérer les évènements d'orientation du téléphone afin de fournir une meilleur UI en fonction des besoins spécifiques.

1. Ouvrez la solution créée dans l'exercice précédent.
2. Afin d'ajouter une gestion automatique de l'orientation des pages, définissez l'attribut *SupportedOrientations* dans l'objet Page.
3. Ouvrez le fichier *Default.xaml* du dossier **About** et modifier la valeur de l'attribut *SupportedOrientations*

```
XAML
SupportedOrientations="PortraitOrLandscape"
```

Remarque : Les propriétés SupportedOrientations pourraient également être modifiées via le fichier de code sous-jacent. Regardez la fonction dans le fichier

MainPage.xaml.cs file pour l'extrait de code en exemple..

4. Compilez et lancez l'application. Allez sur la page « About » et modifiez l'orientation de l'émulateur. Observez le résultat.

Figure 21
La page 'About' en orientation automatique

5. Arrêtez le débogage et retourner au code de l'application.
6. Répétez les étapes 3 et 4 pour les pages suivantes :
   - MainPage.xaml
   - Views\Music\Default.xaml
   - Views\WebBrowser\Default.xaml
   - Views\Video\Default.xaml

   Afin qu'une vidéo affichée par un utilisateur le soit dans un plus grand format lorsque le téléphone est orienté en paysage, utilisez une variante de la procédure ci-dessus pour modifier la page Default.xaml qui se trouve dans le dossier Vidéo.
7. Pour cela, ouvrez le code du fichier Default.xaml.cs dans le dossier Vidéo et ajoutez un gestionnaire pour l'évènement *OrientationChanging* comme ci-dessous :

   (Extrait de Code – *Navigation and Controls – Ex3 Tâche 1 Etape 7 – OrientationChanging event handler*)

C#

```
this.OrientationChanging += (s, e) =>
{
    if (e.Orientation == PageOrientation.Landscape ||
        e.Orientation == PageOrientation.LandscapeLeft ||
        e.Orientation == PageOrientation.LandscapeRight)
    {
        TitleGrid.Visibility = System.Windows.Visibility.Collapsed;
        ContentGrid.SetValue(Grid.RowSpanProperty, 2);
        ContentGrid.SetValue(Grid.RowProperty, 0);
    }
    else
    {
        TitleGrid.Visibility = System.Windows.Visibility.Visible;
        ContentGrid.SetValue(Grid.RowSpanProperty, 1);
        ContentGrid.SetValue(Grid.RowProperty, 1);
    }
};
```

8. Compilez et lancez l'application. Allez sur un des écrans Vidéo et observez la différence d'affichage lorsque l'orientation de l'émulateur change.

Figure 22

Affichage d'une vidéo en mode paysage

9. Arrêtez le débogage et retourner au code de l'application.

Dans cette tâche, vous avez intégré l'évènement OrientationChanging et, dans le cas d'une vidéo, ajouté une fonctionnalité d'augmentation de l'espace disponible sur l'écran pour le MediaElement, permettant à l'utilisateur de visionner une vidéo dans une plus grand format.

Tâche 2 – Gérer la pression du bouton Back

Durant cette tâche, vous apprendrez à gérer l'évènement de pression du bouton « Back » du Windows Phone.

1. Ouvrez le fichier Default.xaml.cs dans le dossier Video.
2. Ajoutez un gestionnaire d'évènement pour l'évènement **BackKeyPress** et stoppez le plaback du MediaElement

   (Extrait de Code – *Navigation and Controls –Ex 3 Tâche 2 Etape 2 – gestionnaire de l'évènement BackKeyPress*)

```
C#
this.BackKeyPress += (s, e) =>
{
   if (media.CurrentState == MediaElementState.Playing)
      media.Stop();
};
```

3. Répétez l'étape précédente pour le fichier Default.xaml.cs dans le dossire *Music*.
4. Compilez et lancez l'application. Allez sur un des écrans Vidéo ou Music et observez l'effet désiré.
5. Arrêtez le débogage et retourner au code de l'application.

Tâche 3 – Ajout d'un Application Bar

Souvent, on veut fournir des fonctionnalités communes à toute ou partie de l'application. Parfois toutefois, on peut souhaiter des fonctionnalités spécifiques sur une page lorsqu'une condition particulière est remplie. Dans d'autres cas, on peut simplement vouloir disposer de plus d'espace sur l'écran. Pour ces différents scénarios, Windows Phone fournit l'Application Bar, qui comprend des boutons et/ou commandes du menu et pouvant être affichée ou masquée. Dans tous les cas, elle occupe un espace fixe sur une petite partie de l'écran, en bas

en mode Portrait, et à gauche ou à droite en mode Paysage (toujours près des boutons du téléphone en tant que tel).

Une classe **ApplicationBar** ajoute des composants d'UI spécifiques, notamment les contrôles **ApplicationBarMenuItem** et **ApplicationBarIconButton**. L'ApplicationBar peut être disposée sur les contrôles **PhoneApplicationPage**. L'AplicationBar et la barre système (en haut de l'écran) peuvent toutes deux être masquées via la propriété **PhoneApplicationPage.FullScreen**.

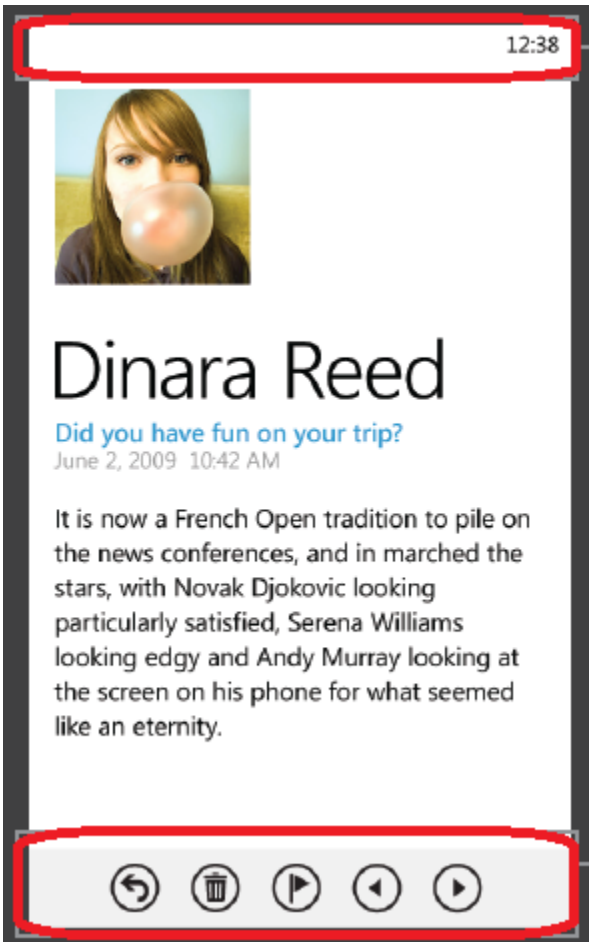

Figure 23

Affichage du telephone

Dans cette section, vous allez :

- Créer des Application Bars en tant que ressources de l'application et les utiliser dans diverses pages
- Ajouter la prise en charge du plein-écran
- Ajouter des boutons à l'Application Bar

1. Ajoutez une référence à l'assembly Microsoft.Phone.Shell

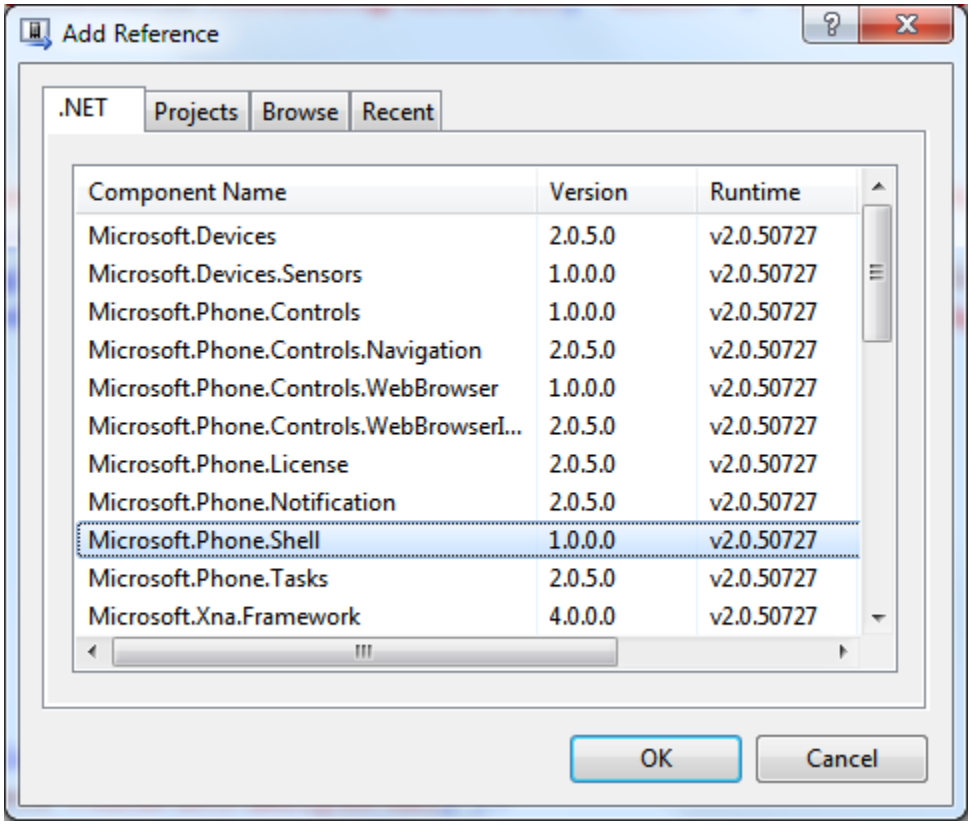

Figure 24

Ajout d'une référence à l'assembly Microsoft.Phone.Shell

2. Ouvrez App.xaml.
3. Ajoutez l'extrait suivant à l'objet de l'application

XAML

```
xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone.Shell"
```

4. Localisez la ressource (ajouté précédemment) **ImagesLocations**, à laquelle vous allez ajouter un certain nombre de ressources.
5. Ajoutez tout d'abord une application bar au l'écran MainMenu. Elle comprendra 2 boutons et 1 menu. Les boutons pointeront vers les Images et écrans WebBrowser (eux-aussi précédemment créés), et le menu ramènera à l'écran « About ». Ajoutez le code suivant dans app.xaml, après ImagesLocation :

(Extrait de Code – *Navigation and Controls – Ex 3 Tâche 3 Etape 5 – MainMenu Application Bar*)

XAML

```
<shell:ApplicationBar x:Key="MainAppBar" Visible="True">
    <shell:ApplicationBar.MenuItems>
        <shell:ApplicationBarMenuItem Text="About"
Click="ApplicationBarAbout_Click" />
    </shell:ApplicationBar.MenuItems>

    <shell:ApplicationBar.Buttons>
        <shell:ApplicationBarIconButton IconUri="ie_icon.png"
Click="ApplicationBarIconWebBrowserButton_Click" />
        <shell:ApplicationBarIconButton IconUri="pictures_Icon.png"
Click="ApplicationBarIconPictures_Click" />
    </shell:ApplicationBar.Buttons>
</shell:ApplicationBar>
```

6. Ouvrez le ficher App.xaml.
7. Ajoutez l'instruction suivante :

C#

```
using Microsoft.Phone.Controls;
```

Lorsque l'on travaille avec l'Application Bar et que l'on fournit des fonctionnalités communes à l'ensemble de l'application, la localisation du code doit naturellement être la classe App du fichier App.xaml.cs. Le code écrit dans cette classe fonctionnera à travers toute l'application, puisque l'objet est créé par le runtime Silverlight à l'initialisation de l'application, et persiste durant tout le cycle de vie de celle-ci. Dans ce cas, le gestionnaire d'évènement pour la fonctionnalité de l'Application Bar est un bon exemple de code lié à l'ensemble de la logique applicative, et non juste à une page spécifique.

8. Ajoutez des gestionnaires d'évènement pour le menu « About » de l'application bar et 2 boutons

(Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 8 – Application Bar event handlers*)

C#

```
private void ApplicationBarIconWebBrowserButton_Click(object sender,
EventArgs e)
```

```
{
    PhoneApplicationFrame root = Application.Current.RootVisual as
PhoneApplicationFrame;
    root.Navigate(new Uri("/WebBrowser/www.bing.com",
UriKind.Relative));
}

private void ApplicationBarIconPictures_Click(object sender, EventArgs
e)
{
    PhoneApplicationFrame root = Application.Current.RootVisual as
PhoneApplicationFrame;
    root.Navigate(new Uri("/Pictures", UriKind.Relative));
}

private void ApplicationBarAbout_Click(object sender, EventArgs e)
{
    PhoneApplicationFrame root = Application.Current.RootVisual as
PhoneApplicationFrame;
    root.Navigate(new Uri("/About", UriKind.Relative));
}
```

9. Ouvrez le fichier MainPage.xaml et supprimez les TextBlocks et Hyperlinks inutiles :

XAML

```
<TextBlock Text="Images" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
<HyperlinkButton NavigateUri="/Pictures" Content="Imges"
Style="{StaticResource PhoneHyperlinkStyle}"/>

<TextBlock Text="Web" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
<HyperlinkButton NavigateUri="/WebBrowser/www.bing.com" Content="Web
Browser" Style="{StaticResource PhoneHyperlinkStyle}"/>

<TextBlock Text="About" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>
<HyperlinkButton NavigateUri="/About" Content="About"
Style="{StaticResource PhoneHyperlinkStyle}"/>
```

10. Ajoutez une référence à la ressource créée à l'objet PhoneApplicationPage de MainMenu :

XAML

```
ApplicationBar="{StaticResource MainAppBar}"
```

11. Compilez et lancez l'application. Vérifiez la navigation en cliquant sur les boutons de la barre d'applications et du menu.

> **Astuce** : Pour accéder au menu cliquez sur le symbole "…" de l'*Application Bar*

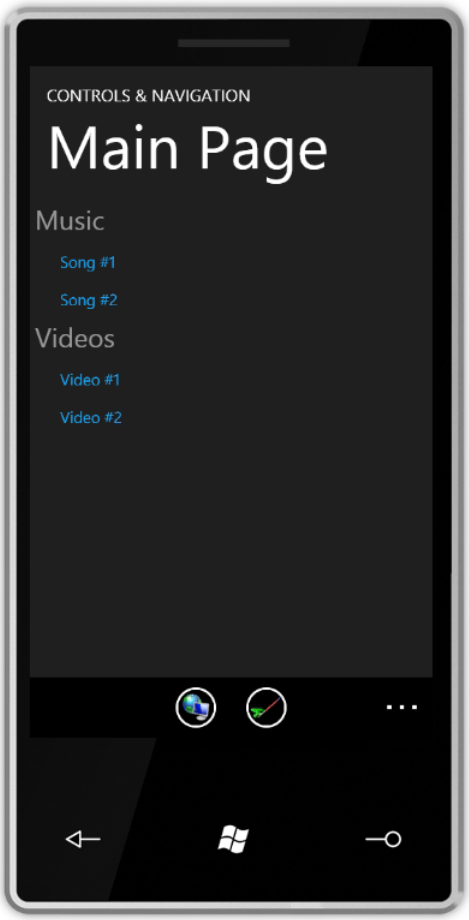

Figure 25
Application avec ApplicationBar

12. Stoppez le débogage et retournez au code de l'application.

13. Afin de masquer l'Application Bar, modifiez la valeur de la propriété **FullScreen** de la page.
14. Ouvrez le fichier MainPage.xaml et ajoutez l'extrait de markup suivant après les liens Video :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 14 – MainPage FullScreen hyperlink*)

    XAML

```
<TextBlock Text="Full Screen" Style="{StaticResource
PhoneTextGroupHeaderStyle}"/>

<HyperlinkButton Content="Toggle Full Screen" Click="OnFullScreenToggle"
Style="{StaticResource PhoneHyperlinkStyle}"/>
```

15. Passez au code de la page et ajoutez un gestionnaire pour l'évènement **Hyperlink Click :**

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 15 – OnFullScreenToggle event handler*)

    C#

```
private void OnFullScreenToggle(object sender, RoutedEventArgs e)
{
    this.FullScreen = !this.FullScreen;
}
```

16. Compilez et lancez l'application. Cliquez sur le lien "Full Screen" et notez la disparition de la barre d'applications et des infos système.

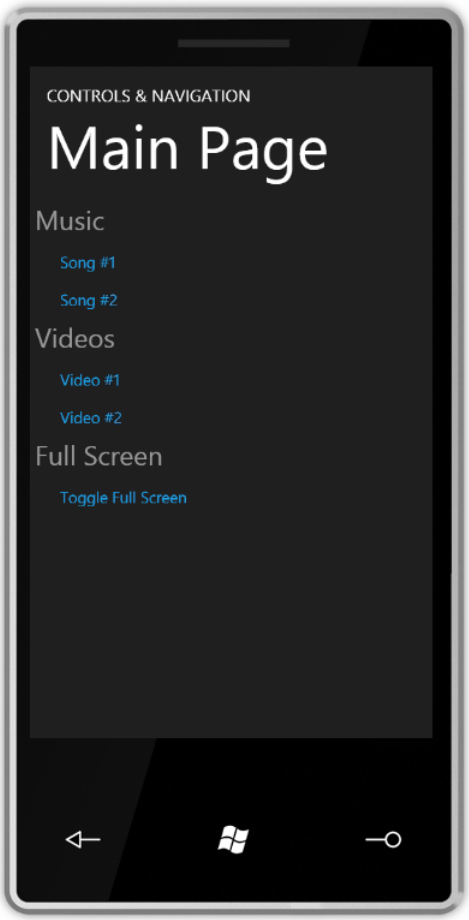

Figure 26
L'application en plein écran

17. Stoppez le débogage et retournez au code de l'application.
18. Ouvrez le fichier App.xaml pour ajouter 2 barres d'applications supplémentaires, l'une pour la page Images, l'autre pour les autres pages.
19. Ajoutez tout d'abord "GlobalAppBar" qui sera utilisée dans toutes les pages (après "MainAppBar") :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 19 – Global Application Bar*)

XAML

```
<shell:ApplicationBar x:Key="GlobalAppBar" Visible="True">
    <shell:ApplicationBar.MenuItems>
```

```
        <shell:ApplicationBarMenuItem Text="About"
Click="ApplicationBarAbout_Click" />
    </shell:ApplicationBar.MenuItems>
</shell:ApplicationBar>
```

20. Ajoutez le markup suivant aux pages ci-dessous :

XAML

```
ApplicationBar="{StaticResource GlobalAppBar}"
```

- Views\Music\Default.xaml
- Views\Video\Default.xaml
- Views\WebBrowser\Default.xaml

21. Ajoutez maintenant une Application Bar pour l'écran Images. Elle devra n'être visible que lorsqu'une image est sélectionnée depuis la ListBox. La barre d'applications fournira 2 éléments, l'un pour afficher l'image sélectionnée en détail et l'autre pour aller vers la page « About ». Ajoutez le markup suivant à App.xaml (après "GlobalAppBar") :

(Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 21 – Picture Application Bar*)

XAML

```
<shell:ApplicationBar x:Key="PictureAppBar" Visible="False">
    <shell:ApplicationBar.MenuItems>
        <shell:ApplicationBarMenuItem x:Name="appBarMenuItemShowPicture"
Text="Show Picture"/>
        <shell:ApplicationBarMenuItem Text="About"
Click="ApplicationBarAbout_Click"/>
    </shell:ApplicationBar.MenuItems>
</shell:ApplicationBar>
```

22. Ouvrez le fichier Default.xaml du dossier *Pictures* et ajoutez-y le code suivant à l'objet PhoneApplicationPage :

XAML

```
ApplicationBar="{StaticResource PictureAppBar}"
```

23. Ouvrez le fichier Default.xaml du dossier *Pictures* et ajoutez-y l'instruction suivante :

```
C#
using Microsoft.Phone.Shell;
```

24. Pour afficher/masque la barre d'applications lorsque l'utilisateur sélectionne une photo, modifies la fonction lstPictures_SelectionChanged comme dans l'extrait de code ci-dessous :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 24 – Pictures Listbox SelectionChanged event handler*)

```
C#
private void lstPictures_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (null != lstPictures.SelectedItem)
    {
        btnRemoveSelection.IsEnabled = true;
        this.ApplicationBar.Visible = true;
    }

    if (photos.Count == 0)
    {
        btnRemoveSelection.IsEnabled = false;
        this.ApplicationBar.Visible = false;
    }
}
```

25. Dans le concepteur de la page, ajoutez l'extrait de code suivant pour souscrire à l'évènement de l'élément du menu de l'Application Bar :

```
C#
(ApplicationBar.MenuItems[0] as ApplicationBarMenuItem).Click += new
EventHandler(ApplicationBar_OnClick);
```

26. Ajoutez la function de gestion d'événement suivant à la classe :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 26 – Application Bar Click event handler*)

```
C#
void ApplicationBar_OnClick(object sender, EventArgs e)
{
```

```
    if (null != lstPictures.SelectedItem)
    {
        PhoneApplicationFrame root = Application.Current.RootVisual as
PhoneApplicationFrame;
        root.Navigate(new Uri("/PictureView/" +
(lstPictures.SelectedItem as Photo).Filename, UriKind.Relative));
    }
}
```

27. Ajoutez une nouvelle page “Landscape Page” au répertoire **\Views\Pictures** et nommez là “PictureView.xaml”.

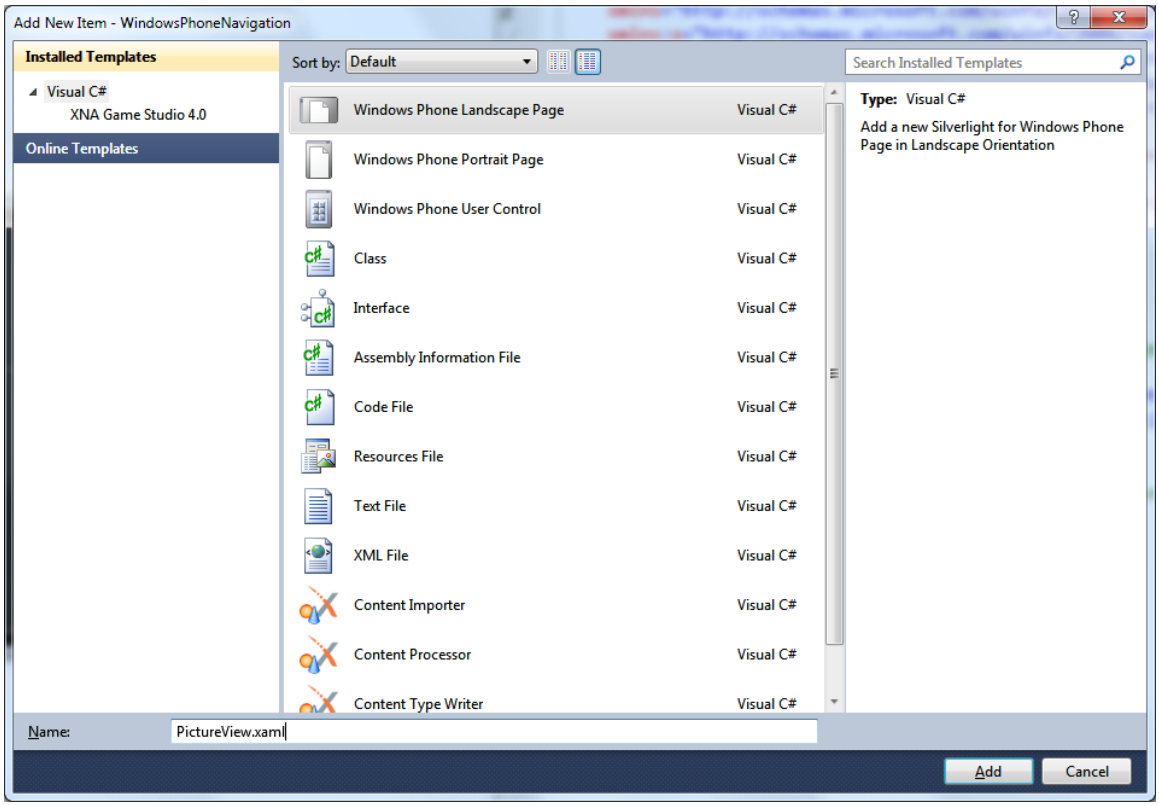

Figure 27

Ajout de la page PictureView.xaml au projet

28. Passez la propriété SupportedOrientation à PortraitOrLandscape :

XAML

```
SupportedOrientations="PortraitOrLandscape"
```

29. Ajoutez “GlobalAppBar” à la page:

XAML

```
ApplicationBar="{StaticResource GlobalAppBar}"
```

30. Modifiez “TitleGrid” comme dans l’extrait de code suivant :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 30 – Picture Page TitleGrid Content*)

XAML

```
<TextBlock x:Name="ApplicationName" Text="{StaticResource AppName}"
Style="{StaticResource PhoneTextPageTitle1Style}"/>
<TextBlock x:Name="ListName" Text="{Binding Filename}"
Style="{StaticResource PhoneTextPageTitle2Style}"/>
```

31. Modifiez “ContentGrid” comme dans l’extrait de code suivant :

XAML

```
<Image Stretch="Uniform" Source="{Binding Image}"/>
```

32. Ouvrez le fichier de code sous-jacent.
33. Ajoutez l’instruction *using* suivante:

C#

```
using WindowsPhoneNavigation.Misc;
```

34. Abonnez vous à l’événement Loaded dans le code de la page :

C#

```
this.Loaded += new RoutedEventHandler(Default_Loaded);
```

35. Créez un gestionnaire d’évènement par défaut. La fonction du gestionnaire d’évènement chargera la photo définie dans les paramètres de navigation et fixera le contexte de donnée pour la page :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 35 – Page Loaded event handler*)

C#

```
void Default_Loaded(object sender, RoutedEventArgs e)
```

```
{
    if (NavigationContext.QueryString.Count > 0)
    {
        Photo thePhoto = new Photo();
        thePhoto.Filename =
NavigationContext.QueryString.Values.First();
        thePhoto.Image =
Utils.GetImage(NavigationContext.QueryString.Values.First());

        this.DataContext = thePhoto;
    }
}
```

36. Ajoutez dans le code un abonnement à l'évènement **OrientationChanging** et affichez/masquez "TitleGrid" en fonction de l'orientation. De plus, modifiez la position de "ContentGrid" en function de l'orientation également. Ajoutez l'extrait de code suivant :

    (Extrait de Code – *Navigation and Controls – Ex3 Tâche 3 Etape 36 – OrientationChanging event handler*)

C#

```
this.OrientationChanging += (s, e) =>
{
    if (e.Orientation == PageOrientation.Landscape ||
       e.Orientation == PageOrientation.LandscapeLeft ||
       e.Orientation == PageOrientation.LandscapeRight)
    {
        TitleGrid.Visibility = System.Windows.Visibility.Collapsed;
        ContentGrid.SetValue(Grid.RowSpanProperty, 2);
        ContentGrid.SetValue(Grid.RowProperty, 0);
    }
    else
    {
        TitleGrid.Visibility = System.Windows.Visibility.Visible;
        ContentGrid.SetValue(Grid.RowSpanProperty, 1);
        ContentGrid.SetValue(Grid.RowProperty, 1);
    }
};
```

37. Compilez et lancez l'application. Naviguez jusqu'à l'écran Images, sélectionnez en une et observez l'apparence de l'Application Bar.

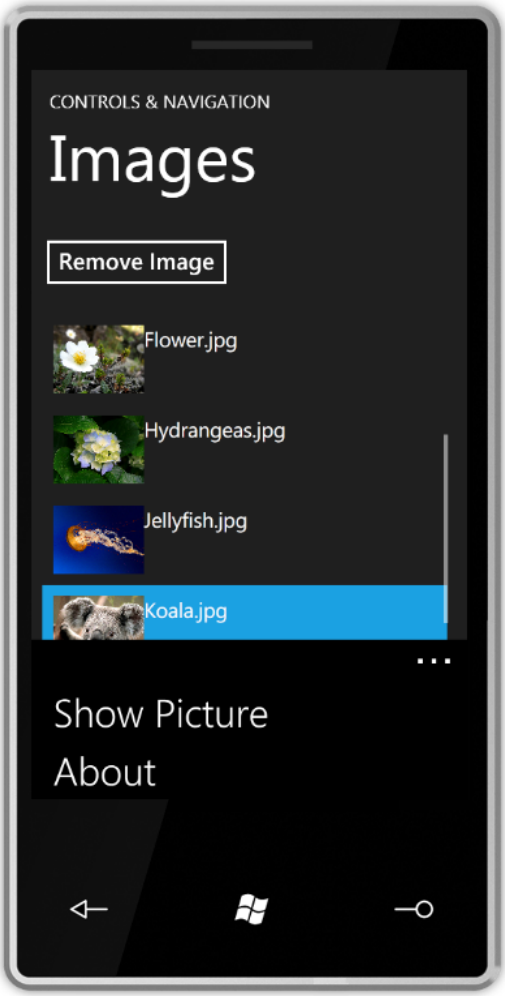

Figure 28

Application Bar avec les elements de menu

38. Cliquez sur une image et vérifiez que la navigation fonctionne :

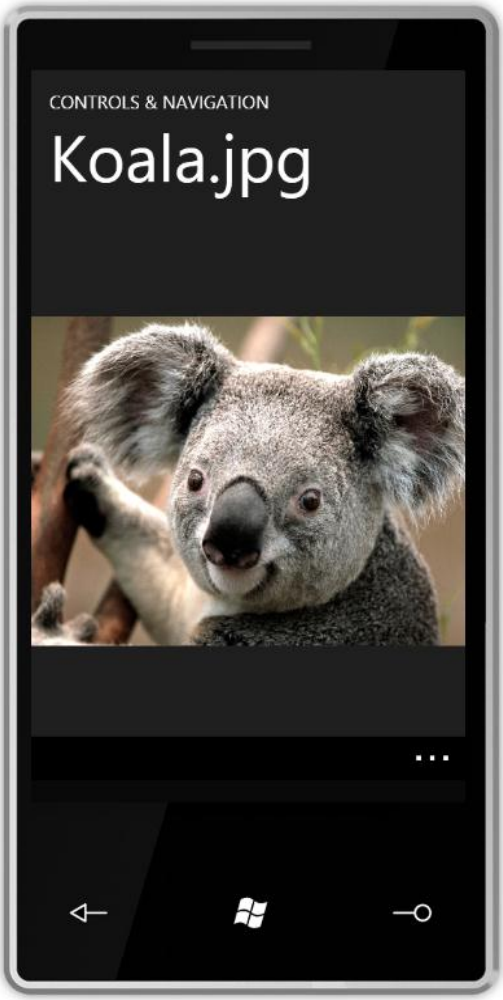

Figure 29

Page PictureView en orientation Portrait

39. Changez l'orientation de l'émulateur pour afficher une image plus importante :

Figure 30

Page PictureView en orientation Paysage

40. Cliquez sur “Back” pour retourner à la liste de photos
41. Sélectionnez quelques photos et cliquez sur le bouton “Remove’. Vérifiez que ces photos sont bien supprimées de la liste.
42. Stoppez le débogage.

Cette étape met fin au 3ème exercice.

Au cours de cet exercice, vous avez créé une Application Bar avec des boutons et des éléments de menu, géré l’orientation des pages, et les évènements du bouton « Back ».

Remarque : La solution pour cet exercice est situé au chemin suivant *{LAB_PATH}\Ex3-IntroductionToWindowsPhoneServices\End*

Résumé

Cet atelier vous a présenté le système d'affichage de Windows Phone et le système de navigation, comprenant les principes de base de la navigation entre différents écrans (pages) dans une application Silverlight pour Windows Phone. Vous avez conçu une application naviguant entre divers écrans, chacun affichant une fonctionnalité du téléphone différente. Vous avez ajouté une Application Bar à la page de navigation principale afin d'exposer des fonctionnalités au sein de l'application. Via ces exercices, vous avez ainsi appris à utiliser Microsoft Visual Phone Developer 2010 Express afin de designer et concevoir une application Windows Phone.