Hands-On Lab

Atelier 2 - Présentation des controles Windows Phone 7

Version : 1.0.0

Dernière mise à jour : 1/10/2012

Vue d'ensemble

Cet atelier vise à présenter le système de rendu de Windows Phone ainsi que la manipulation des contrôles du SDK et du toolkit. Au cours de cet atelier, vous concevrez une application intégrant plusieurs catégories de contrôles pour des scénarios fonctionnels différents. Vous verrez notamment comment utiliser le contrôle **Pivot** pour la séparation de vos écrans. Durant les exercices, vous apprendrez également à utiliser Microsoft Visual Phone Developer 2010 Express pour concevoir et designer une application Windows Phone. Une connaissance de XAML et de Silverlight 4 sont nécessaires 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 :

- Saurez intégrer dans une application Windows Phone des contrôles du SDK ainsi que du toolkit.
- Connaitrez les contrôles disponibles pour les applications Windows Phone comme la ListBox, la LongListSelector, le Pivot, le ToggleSwitch, la ListPicker, etc.

Prérequis

Sont présentés ci-dessous les pré-requis 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 contrôle Pivot et à la création d'écrans intégrés dans ce dernier.
2. Introduction à l'utilisation des contrôles du SDK et du toolkit.

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

Exercice 1 : Introduction au contrôle Pivot et à la création d'écrans intégrés dans ce dernier

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

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".

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

- **Microsoft.Phone.Controls**
- **Microsoft.Phone.Controls.Toolkit**

Le fichier MainPage.xaml, comme tous les éléments **PhoneApplicationPage** créés à partir du modèle d'élément du projet, contient 2 grilles nommées TitlePanel et ContentPanel.

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.

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 **WindowsPhoneApplication** 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**, sélectionner le template « **Windows Phone Application**» puis saisissez le nom « **WindowsPhoneControls** » et cliquez sur **OK**.

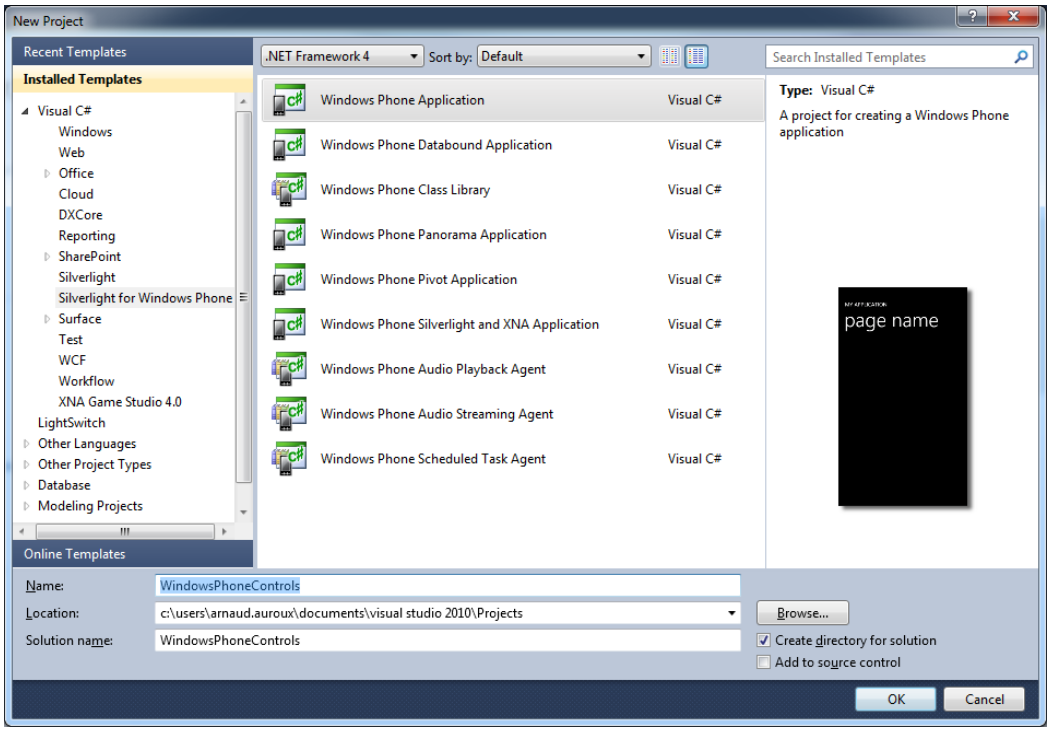


Figure 1

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

4. Une fenêtre vous propose de sélectionner la version de l'OS Windows Phone pour laquelle vous voulez développer votre application, gardez la sélection « Windows Phone OS 7.1 »

5. 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 « **WindowsPhoneControls** ».

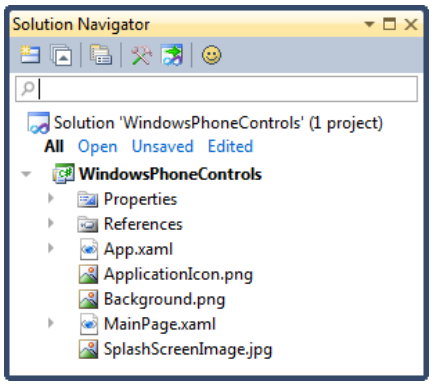


Figure 2

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

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 **WindowsPhoneControls** 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

6. 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 une section **Application.ApplicationLifetimeObjects** permettant de définir des services qui vont être étroitement liés au cycle de vie de l'application. Un service est utilisé par défaut à la création de l'application : **PhoneApplicationService,** il permet d'être notifié des différents changements d'états de l'application. Le code XAML initialise la propriété **RootVisual** de l'**Application** pour définir la première page de celle-ci.

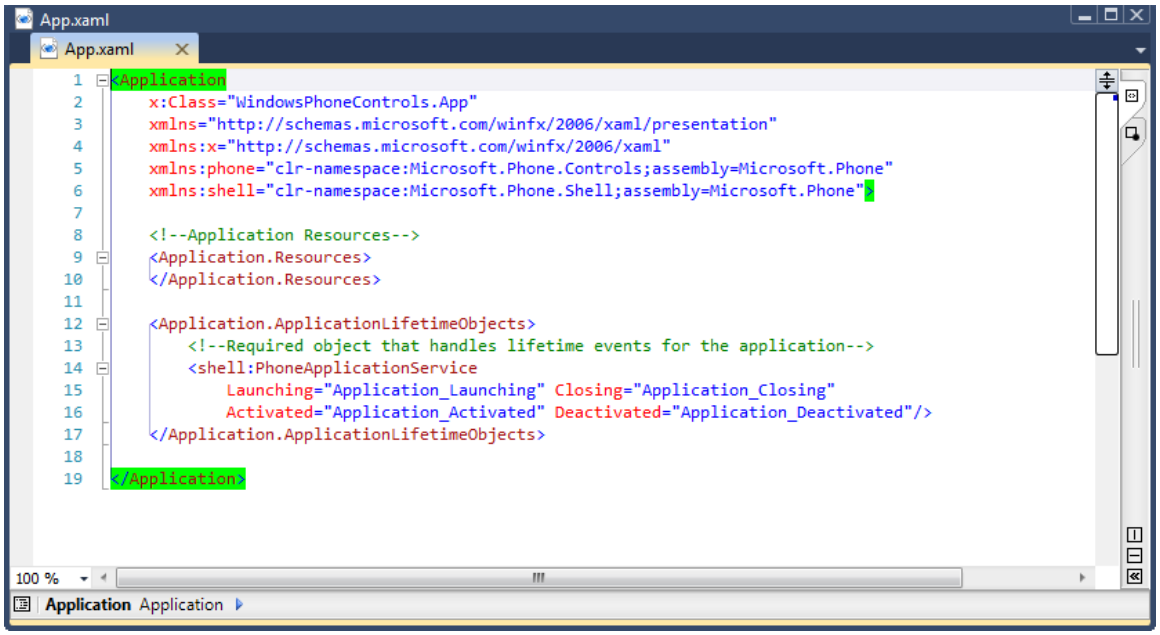


Figure 3

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 peut contenir également des ressources applicatives comme les couleurs, pinceaux (*brush*) et objets de style utilisés au travers de l'application.

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.

7. 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 l'on retrouve les méthodes que l'on avait défini côté XAML pour souscrire aux évènements de changement d'états de l'application (Application_Launching, etc.) et que l'on peut donc définir notre propre logique pour répondre à ces changements. Du code a également été généré pour l'initialisation des différents composants de navigation.

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.

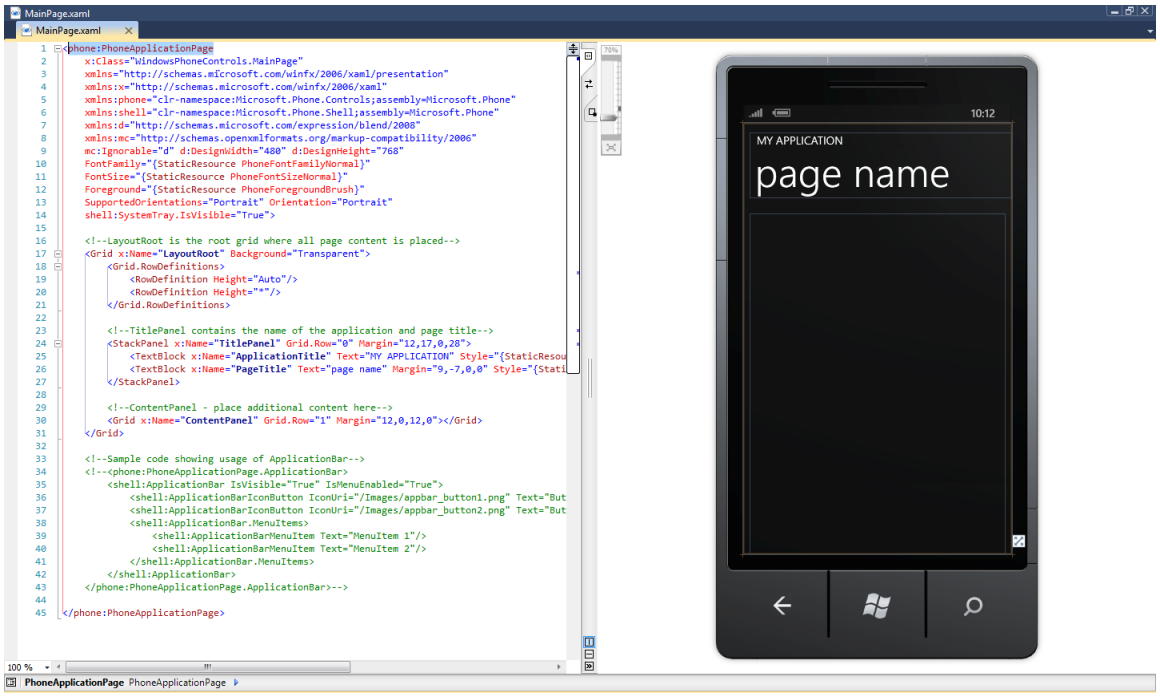


Figure 5

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.

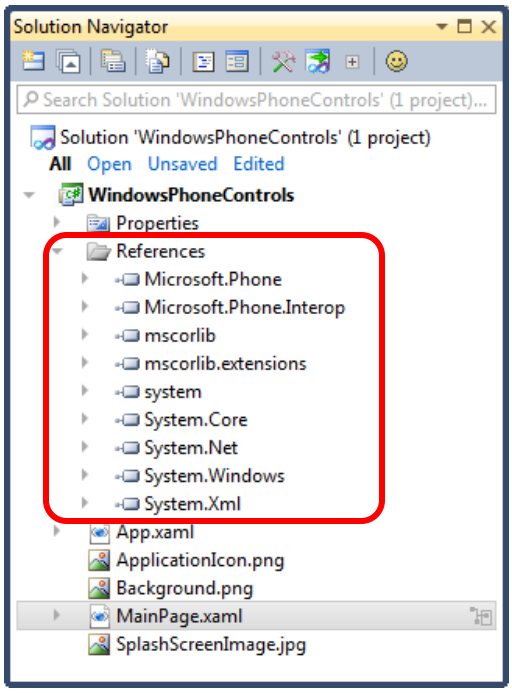


Figure 6

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

10. Pour la mise en œuvre des contrôles de cet atelier, ajoutez une référence vers Microsoft.Phone.Controls et Microsoft.Phone.Controls.Toolkit. Pour cela clic-droit sur le projet puis « Add reference » et sélectionnez les assemblies « Microsoft.Phone.Controls » et « Microsoft.Phone.Controls.Toolkit ».

11. 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.

XAML

```
<system:String x:Key="AppName">CONTROLS</system:String>
```

Le type **String** n'est pas connu par défaut par le moteur XAML, nous allons donc utiliser un namespace XML pour « mapper » un namespace CLR afin de pouvoir utiliser ce type

en XAML. Pour cela ajouter le namespace XML suivant à ceux existant au niveau de la déclaration de la classe Application.

XAML

```
xmlns:system="clr-namespace:System;assembly=mscorlib"
```

12. Ouvrez MainPage.xaml et localisez l'élément nommé TitlePanel (de type StackPanel) et supprimez le.

13. Pour l'intégration du contrôle Pivot, nous devons mapper le namespace CLR qui le contient, pour cela ajoutez aux namespace XML situés en tête du fichier MainPage.xaml le namespace XML suivant :

XAML

```
xmlns:controls="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls"
```

14. Au sein de l'élément Grid nommé « ContentPanel », instanciez le contrôle Pivot :

XAML

```
<Grid x:Name="ContentPanel"
        Grid.Row="1"
        Margin="12,0,12,0">
    <controls:Pivot>

    </controls:Pivot>
</Grid>
```

Le contrôle Pivot va nous permettre de gérer simplement la visualisation et l'accès aux différents écrans que nous allons créer.

15. Nous allons définir le titre du contrôle Pivot, via sa propriété Title, en utilisant la ressource que nous avons défini en étape 11 :

XAML

```
<controls:Pivot Title="{StaticResource AppName}">
```

16. Ajoutez les éléments suivants au Pivot pour nos différents écrans (nous aurons un écran pour gérer des médias, un autre pour gérer un formulaire et enfin un dernier pour la gestion d'une liste de données :

XAML

```
<controls:Pivot Title="{StaticResource AppName}">
    <controls:PivotItem Header="media">
    </controls:PivotItem>
    <controls:PivotItem Header="contacts">
    </controls:PivotItem>
    <controls:PivotItem Header="form">
    </controls:PivotItem>
</controls:Pivot>
```

17. 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 1

Exécution de l'application dans l'émulateur Windows Phone

18. 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.

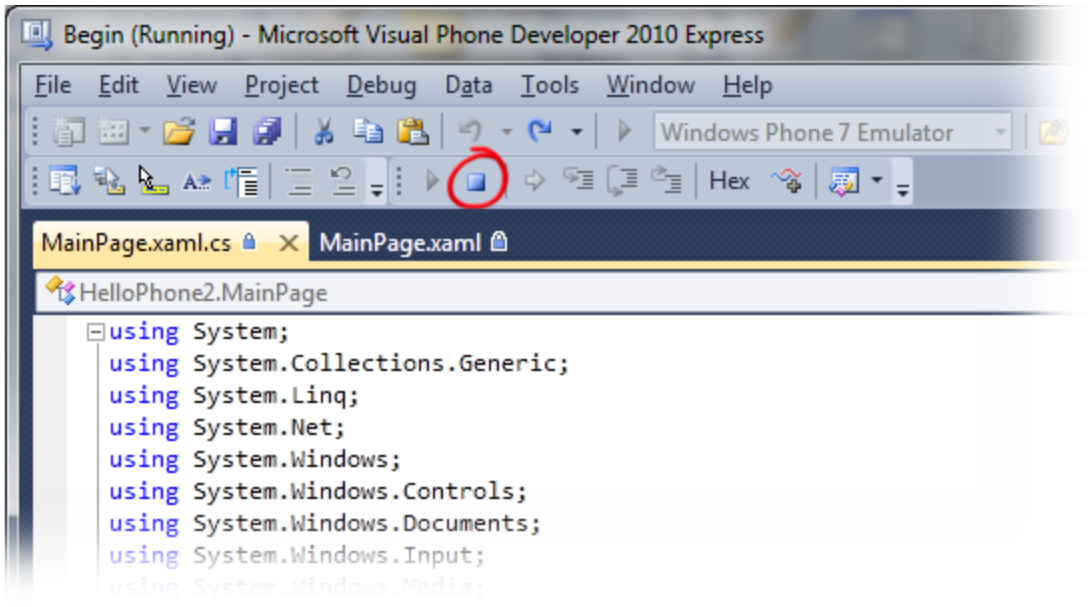


Figure 2

Terminer la session de débogage

3. **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.

19. Ajoutez 3 nouveaux dossiers au projet (Views, Misc, et Assets) pour gérer les éléments à afficher sur les pages du téléphone :
    1. Faîtes un clic droit sur le nom du projet
    2. Sélectionnez Add → New Folder
    3. Nommez **Views** le nouveau dossier
    4. Faîtes de même pour les dossiers **Misc** et **Assets**

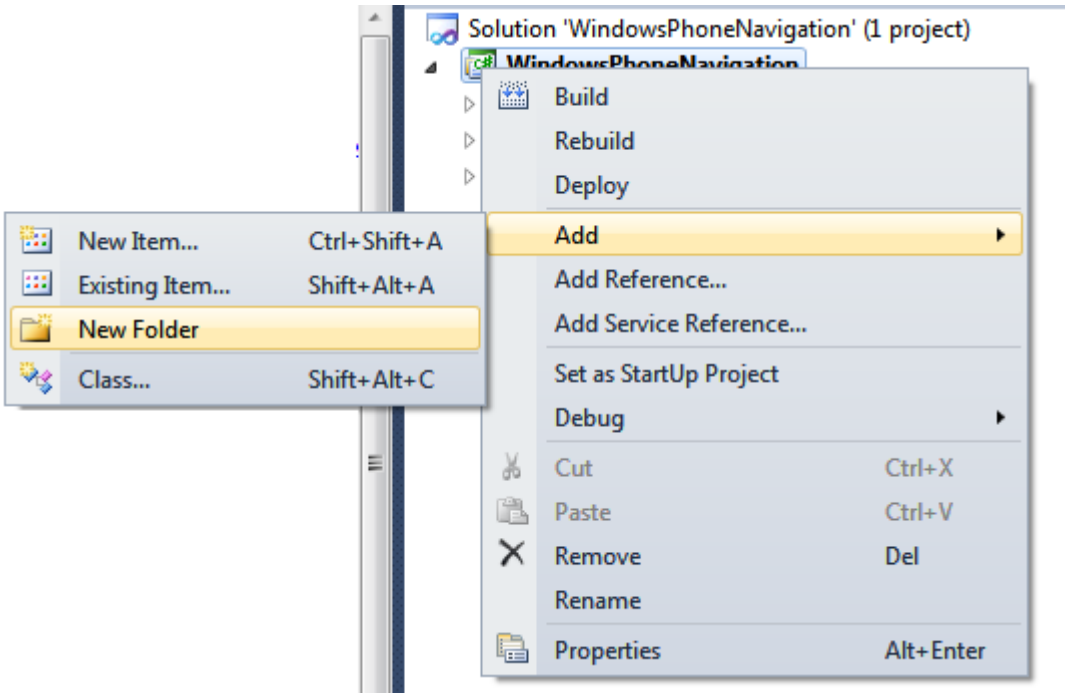


Figure 3
Ajout d'un nouveau dossier au projet

20. Faîtes un clic-droit sur le dossier **Assets** et créez un sous-dossier **Images** ainsi qu'un sous-dossier **Vidéos**. Créez également un sous-dossier **Contact** et **Media** au dossier **Misc**.

> **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.

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

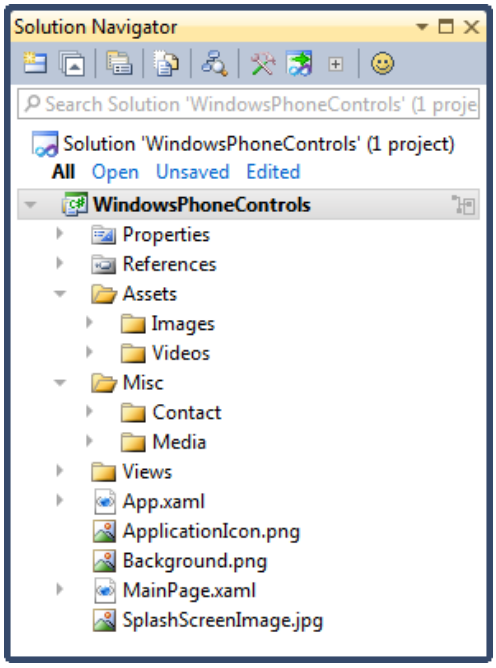


Figure 4

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".

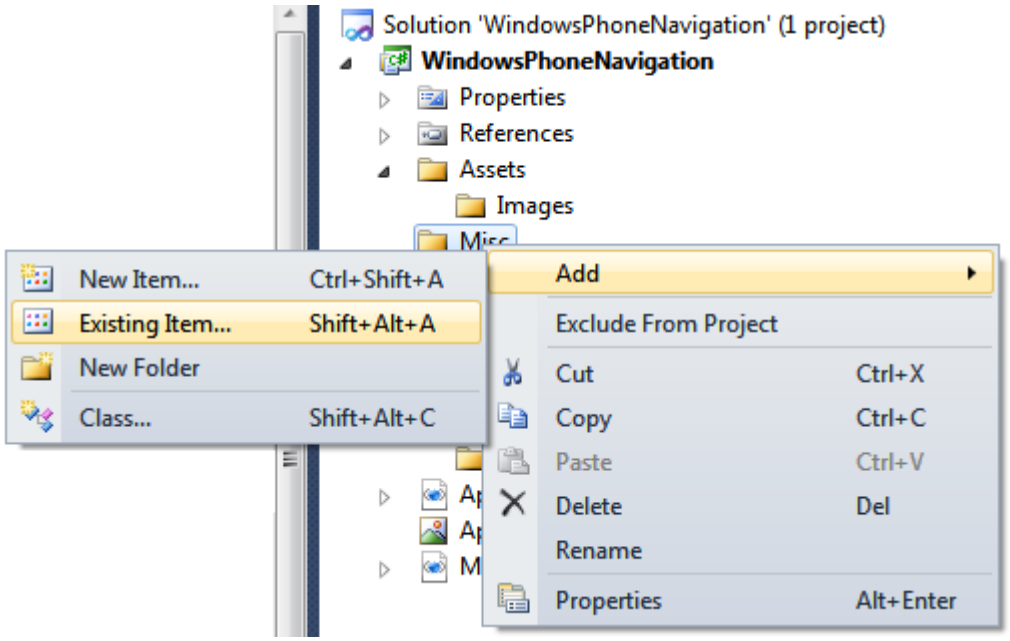


Figure 5

Ajout d'éléments existants aux dossiers du projet

Localisation de l'Asset	Nom de l'Asset	Dossier projet
{LAB_PATH}\Misc\Contact	ContactItem.cs Group.cs	\Misc\Contact
{LAB_PATH}\Misc\Media	MediaItem.c MediaItemType.cs	\Misc\Media
{LAB_PATH}\Assets\Video	Wildlife.wmv	\Assets\Videos
{LAB_PATH}\Assets\Images	Toutes les images	\Assets\Images

22. 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.

23. Passez tous les éléments ajoutés (à l'exception des fichiers portant l'extension « .cs ») en ressources Content :

    - Faites un clic-droit sur le fichier ressource (par exemple, **Desert.jpg**)
    - Sélectionnez **Properties.**

- Sélectionnez **Content**.
- Faites de même pour tous les fichiers de ressources autres que les fichiers .cs

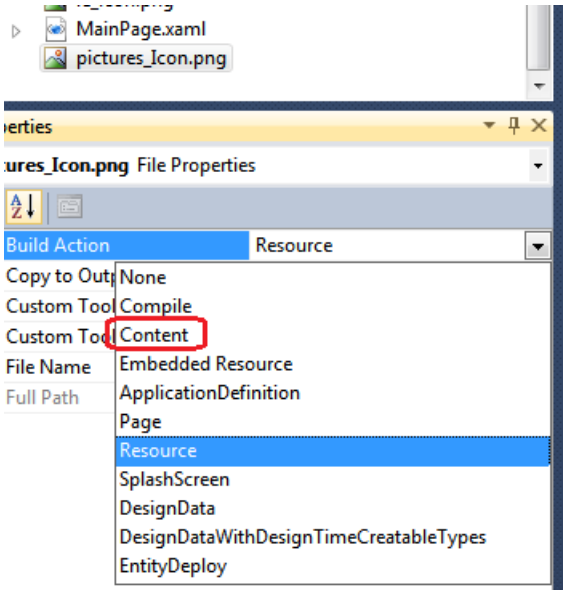


Figure 6

Modification du type de ressource

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

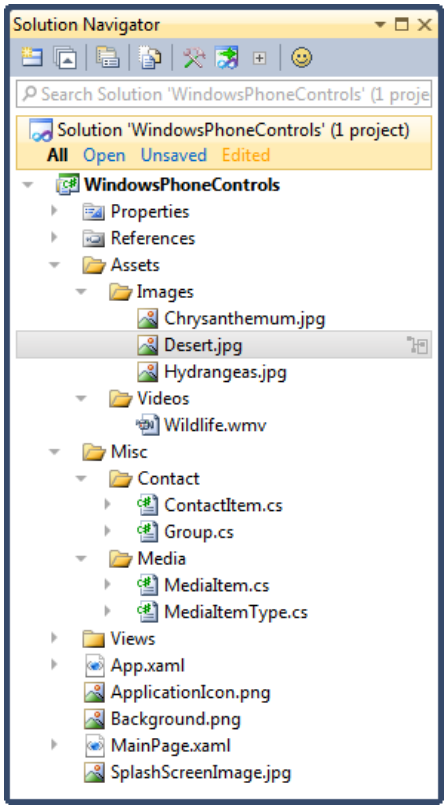


Figure 7

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 écrans au sein du contrôle Pivot

Dans cette tâche, vous ajouterez des écrans et les intégrerez au sein du contrôle Pivot. Ces écrans seront utilisées ultérieurement pour lancer des médias sélectionnés, remplir un formulaire etc.

1. Sous le dossier Views, ajoutez un nouvel élément, **Windows Phone User Control:**
   - Faites un clic-droit sur le dossier **Views**.
   - Sélectionnez **Add** -> **New Item**.
   - Sélectionnez **Windows Phone User Control.**
   - Nommez l'élément **Media.xaml**.

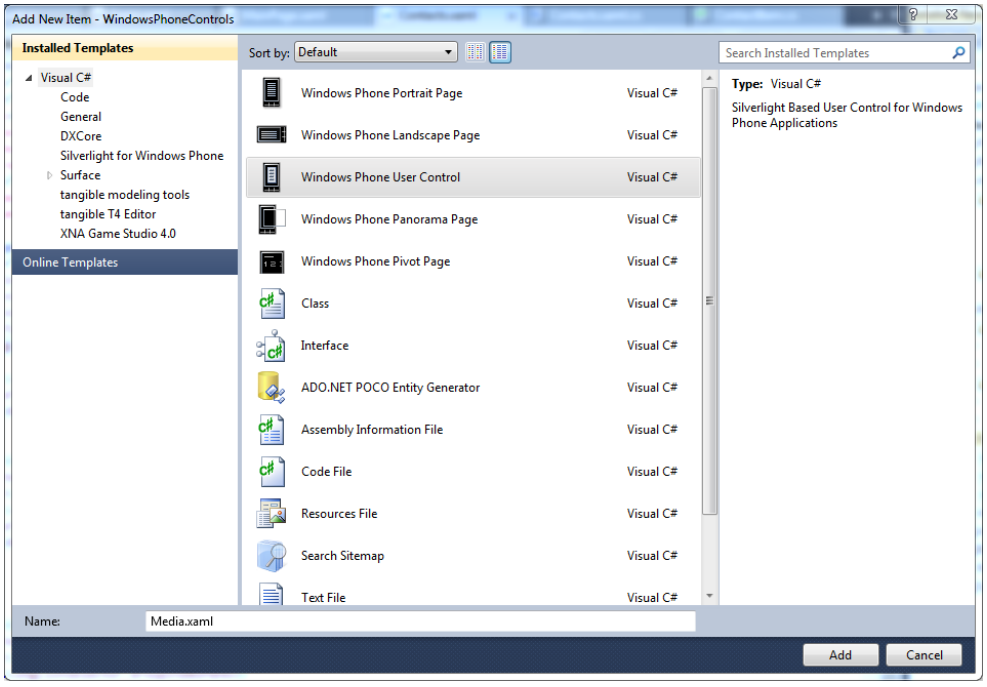


Figure 8

boite de dialogue Add New Item

4. Répétez l'opération pour 2 autres écrans que vous nommerez **Form.xaml** et**Media.xaml**
5. Au niveau de la page MainPage.xaml, nous allons intégrer ces 3 pages au niveau du contrôle Pivot. Pour cela commencez par créer un namespace XML qui permettra d'utiliser les types correspondant aux 3 écrans précédemment créés :

XAML

```
xmlns:local="clr-namespace:WindowsPhoneControls"
```

6. Vous pouvez maintenant incorporer dans les différents éléments du Pivot les 3 écrans :

XAML

```
<controls:Pivot Title="{StaticResource AppName}">
    <controls:PivotItem Header="media">
        <local:Media />
    </controls:PivotItem>
    <controls:PivotItem Header="contacts">
        <local:Contacts />
    </controls:PivotItem>
    <controls:PivotItem Header="form">
        <local:Form />
    </controls:PivotItem>
</controls:Pivot>
```

Ceci conclu l'exercice 1 dans lequel vous avez vu comment mettre en place simplement une application multi-écrans grâce au contrôle Pivot.

Exercice 2: Introduction à l'utilisation des contrôles du SDK et du toolkit.

Dans cette section, vous allez ajouter des fonctionnalités à vos pages media, form et contacts déjà créées grâce aux contrôles du SDK et du Toolkit

Tâche 1 – Ecran média

Au cours de cette tâche, vous allez manipulez le contrôle Listbox destiné à afficher une collection de données sélectionnables qui correspondront à des médias.

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

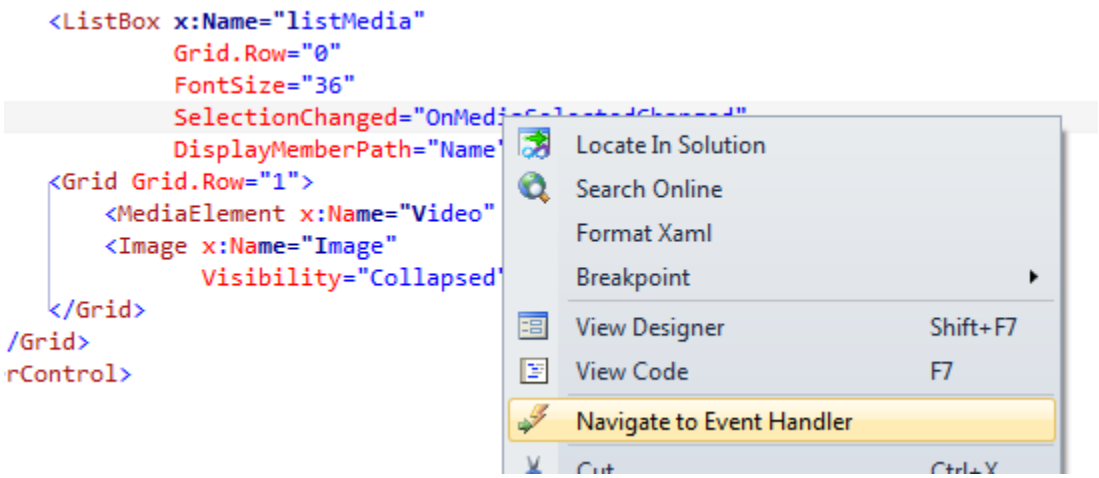
XAML

```
<Grid x:Name="LayoutRoot"
        Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <ListBox x:Name="listMedia"
             Grid.Row="0"
             FontSize="36"
             SelectionChanged="OnMediaSelectionChanged"/>
    <Grid Grid.Row="1">
        <MediaElement x:Name="Video" " Visibility="Collapsed" />
        <Image x:Name="Image" Visibility="Collapsed" />
    </Grid>
</Grid>
```

Ici nous avons donc ajouté 2 lignes à la Grid principale. Nous n'avons pas défini de hauteur donc par défaut ces 2 lignes vont se partager uniformément l'espace vertical disponible. Dans la première ligne, nous avons placé un contrôle ListBox via la propriété **Row** du contrôle Grid qui est une propriété dite « attachée » lui permettant d'être utilisée sur d'autres types de contrôles que le contrôle Grid. Nous avons également

défini une valeur pour la propriété **FontSize** qui sera héritée par les différents éléments enfants de la ListBox destinés à afficher du texte.

Pour gérer la sélection par l'utilisateur d'un élément dans cette liste, nous nous abonnons à l'évènement **SelectionChanged** en définissant un nom pour la méthode à appeler côté *code-behind*, ici « OnMediaSelectionChanged ». Pour générer cette méthode, faites clic-droit sur « OnMediaSelectionChanged » puis « NavigateToEventHandler » :



Dans la deuxième ligne, nous avons ajouté une nouvelle Grid qui contient le contrôle **MediaElement** destiné à jouer des éléments audio ou vidéo et un contrôle **Image.**

Vous pouvez noter que l'on a utilisé la propriété **Visibility** sur ces 2 contrôles avec la valeur « Collapsed ». Ceci a pour conséquence de cacher le contrôle (plus précisément il sera considéré comme hors de l'arbre visuel). Ainsi en fonction du type d'élément sélectionné dans la ListBox, vous pourrez afficher l'un ou l'autre contrôle par code.

Enfin, nous avons nommé plusieurs éléments dans ce code XAML. Cela permettra de pouvoir y accéder côté code-behind.

4. Côté code behind, i.e. dans le fichier Media.xaml.cs, définissez la source de données de la ListBox via sa propriété **ItemSource**. Pour cette source de donnée, nous allons utiliser l'objet métier **MediaItem** qui permettra de définir le nom et la source d'un média et l'énumération MediaItemType pour l'identification du type de média (Image ou Vidéo). Ajoutez le code suivant au constructeur de la classe Media.xaml.cs :

C#

```
this.listMedia.ItemsSource = new List<MediaItem>
{
    new MediaItem
```

```
        {
            Name = "chrysanthemum",
            Source = @"/Assets/Images/Chrysanthemum.jpg",
            MediaItemType = MediaItemType.Image
        },
        new MediaItem
        {
            Name = "desert",
            Source = @"/Assets/Images/Desert.jpg",
            MediaItemType = MediaItemType.Image
        },
        new MediaItem
        {
            Name = "hydrangeas",
            Source = @"/Assets/Images/Hydrangeas.jpg",
            MediaItemType = MediaItemType.Image
        },
        new MediaItem
        {
            Name = "wildlife",
            Source = @"/Assets/Videos/Wildlife.wmv",
            MediaItemType = MediaItemType.Video
        }
    };
```

5.

- Lancer la solution via **F5** et vous devriez avoir le résultat suivant :

En effet le contrôle ListBox ne sachant pas comment afficher l'objet MediaItem (qui est le type de notre source de données), il utilise la représentation sous forme de chaîne de caractères de cet objet. Pour définir la propriété de cet objet que l'on souhaite afficher, nous allons utiliser la propriété **DisplayMemberPath** du contrôle ListBox :

XAML

```
<ListBox x:Name="listMedia"
            Grid.Row="0"
            FontSize="36"
            SelectionChanged="OnMediaSelectedChanged"
            DisplayMemberPath="Name" />
```

On spécifie ici que l'on souhaite utiliser la propriété Name de l'object MediaItem pour l'affichage des données au sein de la ListBox. Relancez la solution :

- Pour gérer la sélection du média, ajoutez le code suivant à la méthode OnMediaSelectedChanged dans le fichier Media.xaml.cs :

C#

```
MediaItem mediaItem = (MediaItem)e.AddedItems[0];
if (mediaItem.MediaItemType == MediaItemType.Image)
```

```
{
    this.Image.Source = new BitmapImage(new System.Uri(mediaItem.Source,
 System.UriKind.Relative));
    this.Image.Visibility = System.Windows.Visibility.Visible;
    this.Video.Visibility = System.Windows.Visibility.Collapsed;
    this.Video.Stop();
}
else
{
    this.Image.Visibility = System.Windows.Visibility.Collapsed;
    this.Video.Visibility = System.Windows.Visibility.Visible;
    this.Video.AutoPlay = true;
    this.Video.Source = new System.Uri(mediaItem.Source, System.UriKind.
RelativeOrAbsolute);
    this.Video.Play();
}
```

Nous récupérons donc l'objet sélectionné et selon son type, on affiche ou on cache le contrôle Image ou MediaElement puis on définit la source du contrôle respectif. Pour le cas du MediaElement, on appel la méthode Play() si l'élément sélectionné est de type vidéo ou Stop() dans le cas contraire.

- Lancez la solution, vous devriez maintenant pouvoir visualiser le média sélectionné :

Tâche 2 – Ecran Form

Durant cette tâche, vous créerez un écran formulaire et vous manipulerez différents contrôles utiles dans ce contexte.

1. Ouvrez la solution créée dans l'exercice précédent
2. Ouvrez le fichier **Form.xaml** depuis le dossier Views
3. Ajoutez le namespace XML suivant qui nous permettra d'utiliser certains éléments du toolkit :

XAML

```
xmlns:toolkit="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Too
lkit"
```

4. Ajoutez l'extrait de code suivant à la Grid « LayoutRoot »:

XAML

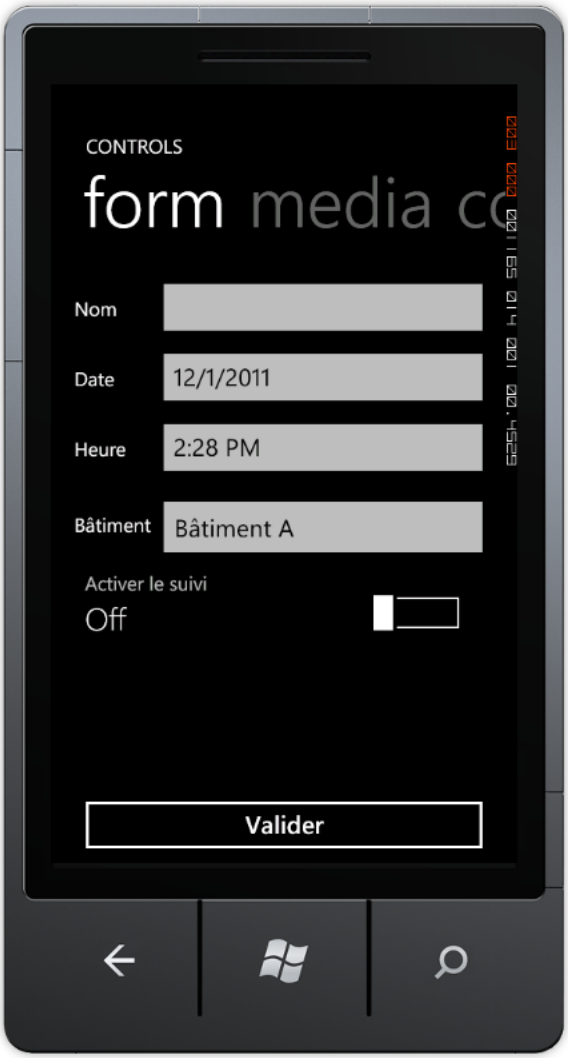
```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition/>
</Grid.ColumnDefinitions>
<TextBlock Text="Nom" HorizontalAlignment="Left" VerticalAlignment="Cent
er"/>
<TextBox Grid.Column="1" x:Name="Name"/>
<TextBlock Text="Date" HorizontalAlignment="Left" VerticalAlignment="Cen
ter" Grid.Row="1"/>
<toolkit:DatePicker Grid.Row="1" Grid.Column="1" x:Name="Date"/>
<TextBlock Text="Heure" HorizontalAlignment="Left" VerticalAlignment="Ce
nter" Grid.Row="2"/>
<toolkit:TimePicker Grid.Row="2" Grid.Column="1" x:Name="Time"/>
<TextBlock Text="Bâtiment" HorizontalAlignment="Left" VerticalAlignment=
"Center" Grid.Row="3"/>
<toolkit:ListPicker Grid.Row="3" Grid.Column="1" x:Name="Building">
    <toolkit:ListPickerItem Content="Bâtiment A"/>
    <toolkit:ListPickerItem Content="Bâtiment B"/>
    <toolkit:ListPickerItem Content="Bâtiment C"/>
    <toolkit:ListPickerItem Content="Bâtiment D"/>
</toolkit:ListPicker>
```

```
<toolkit:ToggleSwitch Header="Activer le suivi" Grid.Row="4" Grid.Column
Span="2" x:Name="FollowUp"/>
<Button Content="Valider" Margin="0,0,0,3"  Grid.Row="5" Grid.ColumnSpan
="2" VerticalAlignment="Bottom" />
```

Ici nous avons donc utilisé les contrôles suivant :

- TextBox (entrée de texte)
- DatePicker (sélection de date)
- TimePicker (sélection d'une heure)
- ListPicker (sélection d'un élément dans une liste)
- ToggleSwitch (définition d'un paramètre binaire)
- Button (lancement d'une action par l'utilisateur)

5. Lancez la solution, vous devriez avoir le rendu suivant :



6. Abonnez-vous à l'évènement Click du bouton « Valider » comme vu précédemment.
7. Dans la méthode liée à cet évènement (ici nommée OnValidateClick) dans le fichier Form.xaml.cs, ajoutez le traitement suivant :

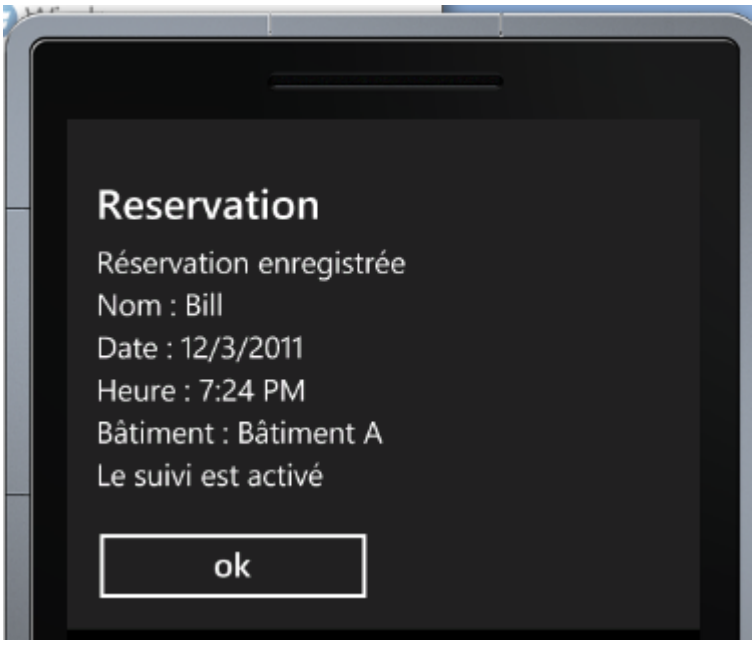
C#

```
private void OnValidateClick(object sender, RoutedEventArgs e)
{
    StringBuilder builder = new StringBuilder();
    builder.Append("Réservation enregistrée")
            .Append(Environment.NewLine)
            .Append("Nom : ")
            .Append(this.Name.Text)
            .Append(Environment.NewLine)
            .Append("Date : ")
            .Append(this.Date.Value.Value.ToShortDateString())
            .Append(Environment.NewLine)
            .Append("Heure : ")
            .Append(this.Time.Value.Value.ToShortTimeString())
            .Append(Environment.NewLine)
            .Append("Bâtiment : ")
            .Append(((ListPickerItem)this.Building.SelectedItem).Content
.ToString())
            .Append(Environment.NewLine)
            .Append("Le suivi ")
            .Append(this.FollowUp.IsChecked.Value ? "est " : "n'est pas
").Append("activé");

    MessageBox.Show(builder.ToString(), "Reservation",
MessageBoxButton.OK);
}
```

Ici nous avons donc récupéré les différentes valeurs saisies par l'utilisateur que l'on formate puis que l'on affiche via le composant **MessageBox**.

8. Lancez la solution, vous devriez avoir le résultat suivant en validant :



Tâche 3 – Ecran Contacts

Au cours de cet atelier nous allons voir comment aller plus loin dans l'affichage d'une collection de données via un contrôle du toolkit : LongListSelector.

1. Ouvrez la solution créée dans l'exercice précédent
2. Ouvrez le fichier **Contacts.xaml** depuis le dossier Vidéo
3. Définissez un namespace XML pointant vers les contrôles du toolkit (comme vue précédemment) ;
4. Ajoutez le code XAML suivant à l'élément racine « LayoutRoot » :

XAML

```
<Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<toolkit:LongListSelector x:Name="ContactsLongList"
                          FontSize="32">
</toolkit:LongListSelector>
<Border VerticalAlignment="Center"
        Padding="10"
        BorderBrush="White"
        BorderThickness="2"
        x:Name="AddContactPanel" Margin="10,0" Visibility="Collapsed" Ba
ckground="Black">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="100" />
        </Grid.RowDefinitions>
        <TextBlock Margin="0,0,1,0"
                   TextWrapping="Wrap"
                   Text="Prénom" Grid.ColumnSpan="2" />
        <TextBox TextWrapping="Wrap"
                   x:Name="FirstName"
                   Grid.Row="1" Grid.ColumnSpan="2" />
        <TextBlock Margin="0,0,2,0"
                   TextWrapping="Wrap"
                   Text="Nom"
                   Grid.Row="2" Grid.ColumnSpan="2" />
        <TextBox TextWrapping="Wrap"
```

```
                    x:Name="Name"
                    Grid.Row="3" Grid.ColumnSpan="2" />
            <Button Content="Enregistrer"
                    Grid.Row="4"
                    Click="OnSaveContactClick" VerticalAlignment="Center" />
            <Button Content="Annuler"
                    Click="OnCancelClick"
                    Grid.Row="4" VerticalAlignment="Center" Grid.Column="1"
/>
    </Grid>
</Border>
<Button Content="Ajouter un contact"
        VerticalAlignment="Top"
        Grid.Row="1"
        Click="OnAddContactClick" />
```

Ici nous avons donc utilisé un contrôle LongListSelector qui va nous permettre de visualiser de façon sophistiquée une liste de contacts. Nous avons également défini plusieurs autres contrôles au sein d'une Grid pour créer un mini-formulaire dans l'écran afin de créer un nouveau contact.

5. Dans le fichier Contacts.xaml.cs, ajoutez le code suivant :

C#

```
private List<ContactItem> contactList = new List<ContactItem>();

private void OnSaveContactClick(object sender, System.Windows.RoutedEventArgs e)
{
    this.contactList.Add(new ContactItem
                            {
                                FirstName = this.FirstName.Text,
                                LastName = this.Name.Text
                            });

    this.AddContactPanel.Visibility = System.Windows.Visibility.Collapsed;

    this.ClearInputs();
}

private void OnAddContactClick(object sender, System.Windows.RoutedEventArgs e)
{
    this.AddContactPanel.Visibility = System.Windows.Visibility.Visible;
}

private void OnCancelClick(object sender, System.Windows.RoutedEventArgs e)
{
    this.AddContactPanel.Visibility = System.Windows.Visibility.Collapsed;
    this.ClearInputs();
}

private void ClearInputs()
```

```
{
    this.FirstName.ClearValue(TextBox.TextProperty);
    this.Name.ClearValue(TextBox.TextProperty);
}
```

Ainsi lors de l'évènement Click sur le bouton « Ajouter un contact » (OnAddContactClick), on va afficher le mini-formulaire (via la propriété Visibility). Lors de l'évènement Click sur le bouton « Annuler », on va simplement le cacher on utilisant la même propriété et on va effectuer une remise à zéro des éléments du formulaire (ClearInputs).
Lors de la validation de l'ajout d'un contact, on va créer un nouvel objet métier « ContactItem » en l'initialisant avec les valeurs entrées par l'utilisateur puis on l'ajoute à la liste des contacts (contactList).

6. Vous allez pouvoir grâce au contrôle LongListSelector grouper les contacts de la liste par ordre alphabétique de leur nom de famille. Pour cela ajoutez la requête suivante à la méthode « OnSaveContactClick » après l'ajout de l'élément ContactItem dans la liste :

C#

```
ContactsLongList.ItemsSource = from contact in contactList
                               group contact by contact.LastName into c
                               orderby c.Key
                               select new Group<ContactItem>(c.Key, c);
```

7. Comme pour la ListBox, nous définissons la collection à afficher par le contrôle LongListSelector via la propriété ItemsSource. Ici nous allons utiliser LINQ (Language Integrated Query) qui est un langage que l'on peut utiliser avec les autres langages supportés par .Net et qui permet d'interroger différents types de sources de données comme ici une liste d'objet. On va donc grouper les contacts par nom de famille et pour chaque groupe nous allons sauvegarder la première lettre du nom de famille ainsi que l'ensemble des contacts dans un objet de type Group.

8. Dans le fichier Contacts.xaml, ajoutez les éléments suivant au contrôle LongListSelector :

XAML

```
<toolkit:LongListSelector.GroupHeaderTemplate>
    <DataTemplate>
        <Border Background="{StaticResource PhoneAccentBrush}">
            <TextBlock Text="{Binding Name}"
                       Margin="5,0,0,0"/>
        </Border>
    </DataTemplate>
</toolkit:LongListSelector.GroupHeaderTemplate>
<toolkit:LongListSelector.GroupItemTemplate>
    <DataTemplate>
        <Border Background="{StaticResource PhoneAccentBrush}"
                Margin="5"
                Padding="5">
            <TextBlock Text="{Binding Name}"
                       Style="{StaticResource PhoneTextLargeStyle}" />
        </Border>
    </DataTemplate>
</toolkit:LongListSelector.GroupItemTemplate>
<toolkit:LongListSelector.ItemTemplate>
    <DataTemplate>
        <StackPanel Orientation="Horizontal"
                    Margin="5,0,0,0">
            <TextBlock Text="{Binding FirstName}" />
            <TextBlock Text="{Binding LastName}"
                       Margin="10,0,0,0" />
        </StackPanel>
    </DataTemplate>
</toolkit:LongListSelector.ItemTemplate>
```

Ici nous avons défini comment le contrôle doit afficher les groupes (GroupItemTemplate), l'en-tête des groupes (GroupItemTemplate) ainsi que les éléments des groupes (ItemTemplate). Notez que nous avons tiré parti des ressources liées au thème graphique courant du système (PhoneAccentBrush, etc.) permettant ainsi à notre application de s'adapter aux préférences utilisateurs.

9. Lancez la solution, vous devriez avoir les résultats suivant lors de l'ajout de plusieurs contacts :

Résumé

Cet atelier vous a présenté l'utilisation basique des contrôles au sein d'une application Windows Phone et vous avez manipulé plusieurs des différents contrôles disponibles à la fois dans le SDK mais également dans le toolkit. Vous avez conçu une application composée de plusieurs écrans grâce au contrôle Pivot et vous avez appris comment interagir avec les différents contrôles qui les composaient. Via ces exercices, vous avez ainsi appris à utiliser les contrôles Silverlight avec Microsoft Visual Studio Developper 2010 Express.