Hands-On Lab

Modèle d'exécution et navigation sous Windows Phone 7

Version: 1.0.0

Dernière mise à jour : 1/10/2012

Sommaire

Vue d'ensemble

Après avoir passé les différents contrôles disponibles dans le second atelier, vous allez maintenant découvrir le modèle d'exécution ainsi que son modèle de navigation ce qui vous permettre de créer des applications plus réactives et plus agréables à l'utilisation. Enfin, vous verrez comment utiliser les API propres à WP7 et comment intégrer votre application dans ses hubs.

Objectifs

A la fin de cet atelier, vous aurez appris :

- Le modèle d'exécution d'une application Windows Phone 7
- Comment créer un service s'exécutant en tâche de fond
- Le modèle de navigation d'une application Windows Phone 7
- Comment utiliser les services de Windows Phone 7
- Comment utiliser les différentes tâches du téléphone
- Comment intégrer son application dans les hubs du téléphone

Pré requis

Afin de mener à bien cet atelier, vous devez installer les éléments suivants :

- Microsoft Visual Studio 2010 Express pour Windows Phone 7

Exercices

Cet atelier est divisé en plusieurs exercices. Ceux-ci sont listés ci-dessous :

1. Présentation du modèle d'exécution
2. Création d'un background agent
3. Présentation du modèle de navigation
4. Introduction aux services Windows Phone
5. Utilisation des tâches
6. Intégration aux hubs du téléphone

Durée estimée pour terminer cet atelier: **180 minutes**.

Atelier : Modèle d'exécution et navigation sous Windows Phone 7

Exercice 1 – Présentation du modèle d'exécution

Dans cet atelier, vous allez découvrir les différents états que peuvent traverser les applications développées pour Windows Phone 7 et vous découvrirez les concepts qui se cachent derrière les termes « Tombstonning » ou encore « FAS ».

1. Créez un nouveu projet de type « Windows Phone Application » de la catégorie « Silverlight for Windows Phone ».
2. Le projet nouvellement créé contient déjà plusieurs fichiers donc « App.xaml » et « App.xaml.cs ». Ces deux fichiers représetent, via le mécanisme de classe partielle, la classe **App** qui dérive de la classe **Application**.
   Cette classe est la racine de votre application : elle contiendra tous autres éléments visuels qui composeront les pages de l'application, elle permet de définir des ressources qui seront accessibles dans toutes ces pages, et le point le plus important ici, elle contient des méthodes branchées aux différents évènements du cycle de vie de l'application.

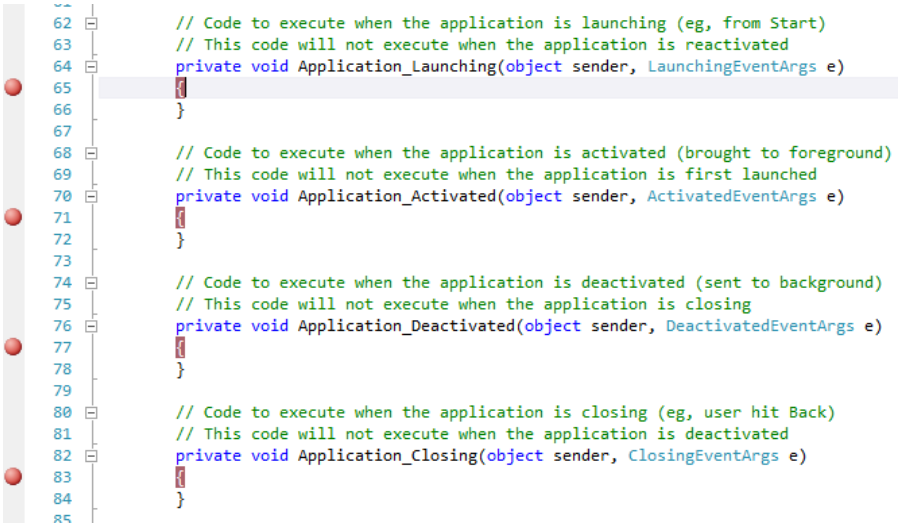
   Le démarrage de l'application a lieu dès que l'utilisateur tape la tuile correspondante dans la liste des applications installées sur son téléphone, ou dès qu'il tape une notification envoyée par l'application.

   Il a ensuite deux cas de figures :

   - L'utilisateur utilise le bouton **Back** du téléphone. Les différentes pages de l'application qui avaient été placées dans le journal de navigation défilent. Une fois l'histoire de navigation totalement dépilé, l'application est quittée et l'utilisateur ramené à l'accueil de son téléphone.
     A présent, il ne peut pas retourner dans l'application en se servant du bouton **Back** de son téléphone. S'il veut l'utiliser à nouveau, il devra taper à nouveau la tuile correspondante ou une notification.
   - L'utilisateur utilise le bouton **Start** du téléphone, utilise un **Chooser** (pour récupérer un contact dans le carnet d'adresse du téléphone ou une image dans une gallerie par exemple) ou que le téléphone est vérouillé et que l'application n'est pas configurée pour continuer à être exécutée dans ce cas précis. Alors, elle est mise en arrière plan, cela signifie qu'elle est désactivée. Elle reste alors en mémoire mais les différents threads qu'elle utilise ne sont plus exécutés. Elle est comme figée. L'utilisateur est amené sur l'écran d'accueil du téléphone.
     S'il appuie sur le bouton **Back**, l'application est restaurée, elle est réactivée dans le même état.
     L'utilisateur peut lancer d'autres applications et les désactiver à leur tour. En gardant le bouton **Back** du téléphone enfoncé, il peut alors accéder aux différentes applications mises en arrière plan. Il s'agit du « FAS », le Fast App Switching.

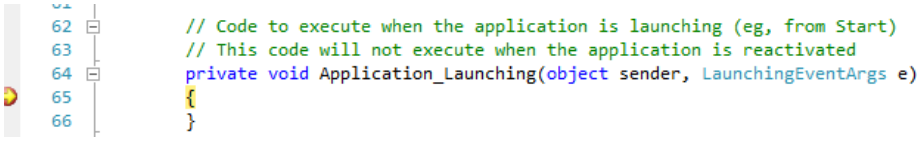
Quand le système d'exploitation du téléphone détectera que la mémoire du téléphone est insuffisante pour exécuter correctement une nouvelle application, il quittera automatiquement l'application qui avait été désactivée en premier. L'état de cette application est alors perdu. Il s'agit du tombstonning.

3. Placez des points d'arrêts dans les méthodes ci dessous (en pressant F9 sur la ligne où le point d'arrêt doit être inséré).

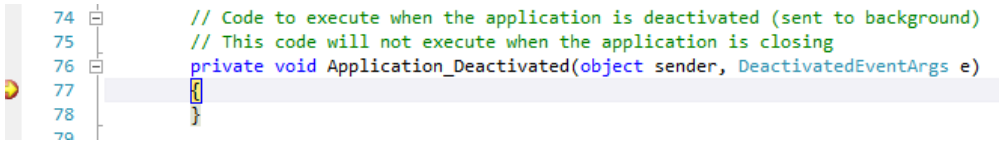


Executez l'application.

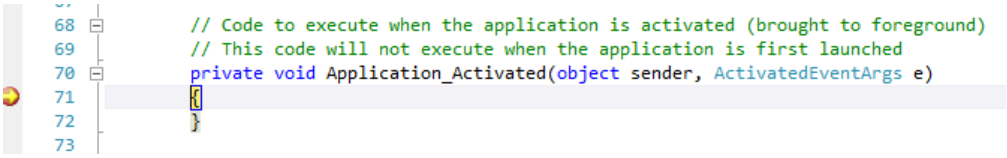
Le debuggueur commence par s'arrêter dans la méthode **Application_Launching**. Le code que vous y insérerez sera donc exécuté au premier lancement de l'application.



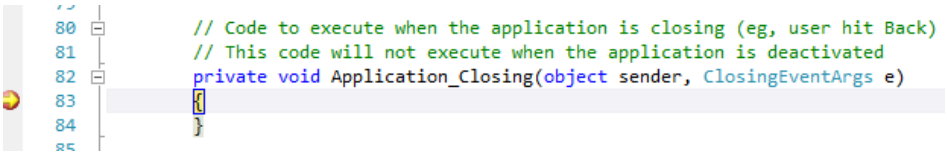
Appuyez sur le bouton **Start** du téléphone. L'application est mise en arrière plan, le debuggueur s'arrête dans la méthode **Application_Deactivated**.



Vous êtes à présent revenu sur l'écran d'accueil du telephone. Appuyez sur la touche **Back**. L'application est réactivée, le debuggueur s'arrête dans la méthode **Application_Activated**.



Pour terminer, appuyez à nouveau sur le bouton **Back** du téléphone. L'application se ferme, le debuggueur passe alors dans la méthode **Application_Closing**.



D'une manière générale, il est recommandé de ne pas faire de traitement trop long des ces méthodes afin de ne pas dégrader l'expérience utilisateur. Pour les évènements de lancement / activation de l'application, déportez vos méthodes d'initialisation et de chargement de données dans un thread en arrière plan. De même, n'attendez pas que les évènements de cloture / désactivation de l'application soient levés, sauvegardez les données qui doivent l'être au fur et à mesure du déroulement du cycle de vie.

4. Pour bien supporter le Fast App Switching dans son application, il est possible d'utiliser la propriété **IsApplicationInstancePreserved** de l'objet de type **ActivatedEventArgs** dans la méthode **Application_Activated**. Cette propriété permet de savoir lors de l'activation de l'application si celle-ci était préservée en arrière plan ou non.
   Modifiez la méthode **Application_Activated** de votre projet et insérez un point d'arrêt dans la condition.

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
    if (e.IsApplicationInstancePreserved)
    {
    }
}
```

Démarrez l'application. Appuyez sur le bouton **Start** du téléphone. Une fois revenu sur le menu principal, appuyez sur le bouton **Back** du téléphone. Le debuggueur s'arrête alors dans la condition que vous venez d'ajouter à la méthode **Application_Activated**.

Grâce à Visual Studio, il est possible de simuler le tombstonning de votre application. Faîtes un clic droit sur votre projet et cliquez sur « Properties ». Dans l'onglet « Debug », cochez la case « Tombstone upon deactivation while debugging ». Sauvegardez.

A nouveau, démarrez l'application. Appuyez sur le bouton **Start**, puis sur le bouton **Back**. Le debuggueur ne s'arrête plus sur le point d'arrêt que vous aviez défini. En effet, l'application n'était plus en arrière-plan.

Ajoutez un contrôle de type **TextBox** à la page de votre application. Répétez les deux scénarios précédents (avec le tombstonning automatique et sans), et à l'exécution de votre application, insérez du texte dans le contrôle.

- Dans le cas ou l'application est mise en arrière plan, lorsqu'elle est restaurée, le contenu de la **TextBox** l'est également.
- Dans le cas du tombstonning, lorsque l'application est restaurée, le contenu de la **TextBox** est perdu.

Chaque page possède une propriété **State** de type **Dictionnary<string, object>**. Cette propriété est restaurée lorsque l'application est réactivée suite à un tombstonning, il est donc possible d'y stocker le status de contrôles afin que l'utilisateur retrouve son application telle qu'il l'avait laissée.

Exercice 2 – Création d'un background agent

Dans cet exercice, vous allez découvrir les techniques utilisables pour que du code de votre application soit exécuté alors que celle-ci n'est plus au premier plan.

Il existe deux méchanismes capables d'effectuer un tel traitement :

- Les **PeriodicTask**. En règle générale, elles sont répétées toutes les 30 minutes pour une durée de 25 secondes. Cependant, certains facteurs tels que l'état de la batterie ou de la connexion réseau peuvent influer sur ce déroulement.
- Les **ResourceIntensiveTask**. Elles peuvent s'exécuter pendant un laps de temps plus long dès lors que plusieurs conditions sont remplies : activité du processeur du téléphone, téléphone branché à une source d'énergie, connexion au réseau de bonne qualitée.

Dans les deux cas, la mémoire vive consommée par la tâche ne devra pas dépasser 6 MB. Il est possible de définir une date d'expiration pour les agents, la période maximale étant de deux semaines. A la fin de cette période, votre application devra à nouveau enregistrer les tâches.

1. Commencez par créer un projet de type « Windows Phone Application » dans la catégorie « Silverlight for Windows Phone ».

2. Ajoutez un projet de type « Windows Phone Scheduled Task Agent ». Ce projet va héberger un agent capable de traiter une tâche de type **PeriodicTask** et/ou une tâche de type **ResourceIntensiveTask**. Il ne peut y avoir qu'un seul Scheduled Task Agent par application Windows Phone 7.

3. Dans le projet « Windows Phone Application », ajoutez une référence vers le projet « Windows Phone Scheduled Task Agent ». En plus d'ajouter la référence, le fichier WMAppManifest.xml est édité et une nouvelle section est ajoutée. Elle permet de définir le Scheduled Task Agent lié à l'application.

```
  <Tasks>
    <DefaultTask Name="_default" NavigationPage="MainPage.xaml" />
    <ExtendedTask Name="BackgroundTask">
      <BackgroundServiceAgent Specifier="ScheduledTaskAgent" Name="CoachSch
eduledTaskAgent" Source="CoachScheduledTaskAgent" Type="CoachScheduledTaskA
gent.ScheduledAgent" />
    </ExtendedTask>
  </Tasks>
```

4. Dès qu'une tâche doit être exécutée, la méthode de **OnInvoke** de l'agent est appelée en recevant la tâche en paramètre. Grâce à l'opération **is**, il est possible de savoir s'il s'agit d'une **PeriodicTask** ou d'une **ResourceIntensiveTask**.

```
if (task is PeriodicTask)
{
    // cas d'une PeriodicTask
```

```
}
else
{
    // cas d'une ResourceIntensiveTask
}
```

5. Lors de l'exécution d'une tâche, il est possible d'envoyer des notifications à l'utilisateur en utilisant la classe **ShellToast**. Une notification de type Toast ne sera pas affichée si l'application qui en est à l'origine est actuellement au premier plan. Il est également possible de renseigner la propriété **NavigationUri** avec le chemin vers une page précise de l'application, et éventuellement d'y ajouter des **QueryStrings**. Enfin, utilisez la méthode **Show** pour afficher la notification.

```
protected override void OnInvoke(ScheduledTask task)
{
    ShellToast popupMessage = new ShellToast()
    {
        Title = "Ma tâche",
        Content = "Ma tâche a été exécutée",
        NavigationUri = new Uri("/MainPage.xaml", UriKind.Relative)
    };

    popupMessage.Show();

    NotifyComplete();
}
```

6. Au sein de votre application Windows Phone classique, l'administration des tâches à exécuter par l'agent se fait au travers de la classe **ScheduledActionService** et ses méthodes statiques.
   Notez que les tâches doivent être nommées : c'est un point très important. En effet, il est possible de récupérer une tâche auprès du ScheduledActionService en utilisant la méthode **Find**. L'objet renvoyé par cette méthode est de type **ScheduledAction**, classe de base des classes **PeriodicTask** et **ResourceIntensiveTask**. Notez également quelle doivent également contenir une description ! Celle-ci sera disponible dans les paramètres du téléphone de l'utilisateur, dans la liste des tâches d'arrière plan.
   L'ajout et la suppresion de tâche se fait en utilisant respectivement les méthodes **Add** et **Remove**.

   Enfin, la classe **ScheduledActionService** possède une méthode **LaunchForTest**, vous permettant, à des fins de debugguage, de planifier le lancement d'une tâche sans avoir à attendre qu'elle soit appelée selon le cycle classique.

   Modifiez le constructeur de la classe **MainPage** de votre application comme ci-dessous.

```
public MainPage()
{
    InitializeComponent();
```

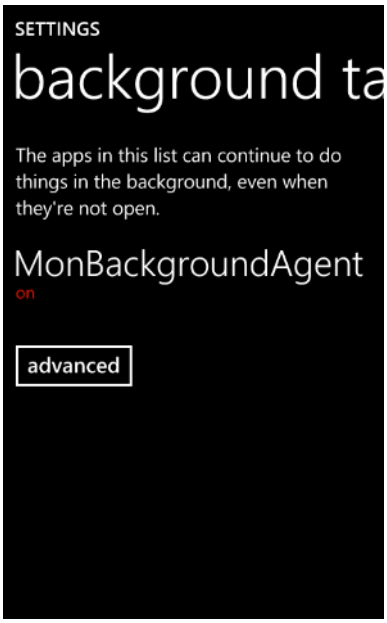
```
    var task = ScheduledActionService.Find("MaTache") as PeriodicTask;

    if (task != null)
        ScheduledActionService.Remove("MaTache");

    if (task == null)
    {
        task = new PeriodicTask("MaTache");

        task.Description = "Ma tâche";

        ScheduledActionService.Add(task);
        ScheduledActionService.LaunchForTest(task.Name, TimeSpan.FromSecond
s(20));
    }
}
```

7. Exécutez l'application. Quand elle s'affiche, appuyez sur le bouton **Start** du téléphone et attendez quelques secondes.

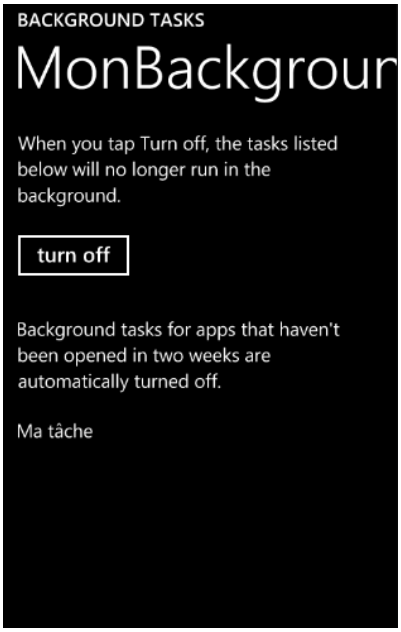
   La barre de status de Visual Studio affiche alors le message : « Launching Background Task ».

   Et la nofication est affichée sur l'écran du téléphone.

   En vous rendant les paramètres de votre téléphone, dans l'onglet Applications et l'élément Tâches d'arrière plan, vous retrouverez votre application.



En sélectionnant votre application, l'utilisateur voit alors la description de la tâche et peut choisir de désactiver l'agent d'exécution de votre application.



Exercice 3 – Présentation du modèle de navigation

Dans cette section, nous allons créer un nouveau projet. Une fois cela fait, nous allons :

- 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;orderDat
e={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.

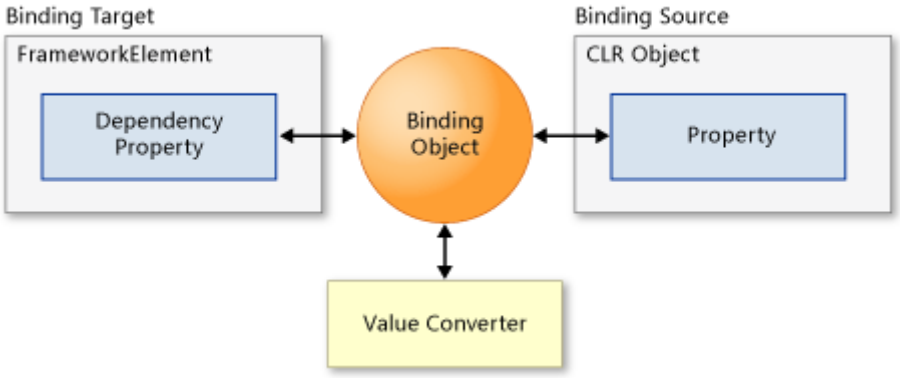


Figure 1
Data binding

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**.

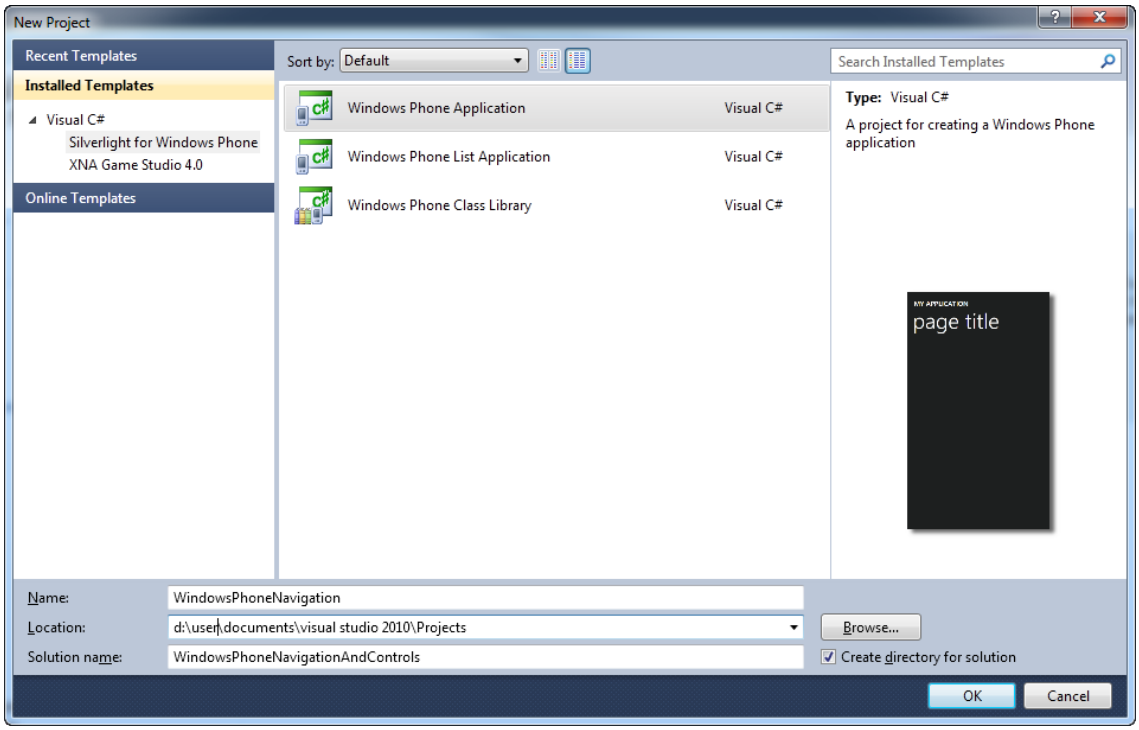


Figure 1

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** ».

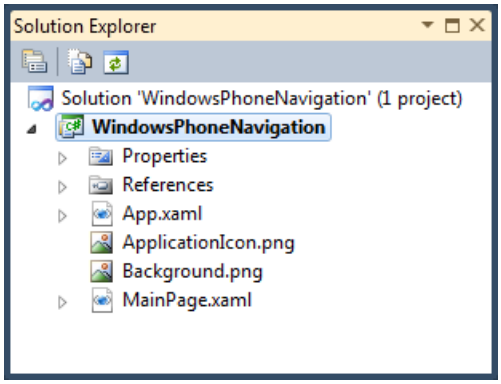


Figure 2

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.

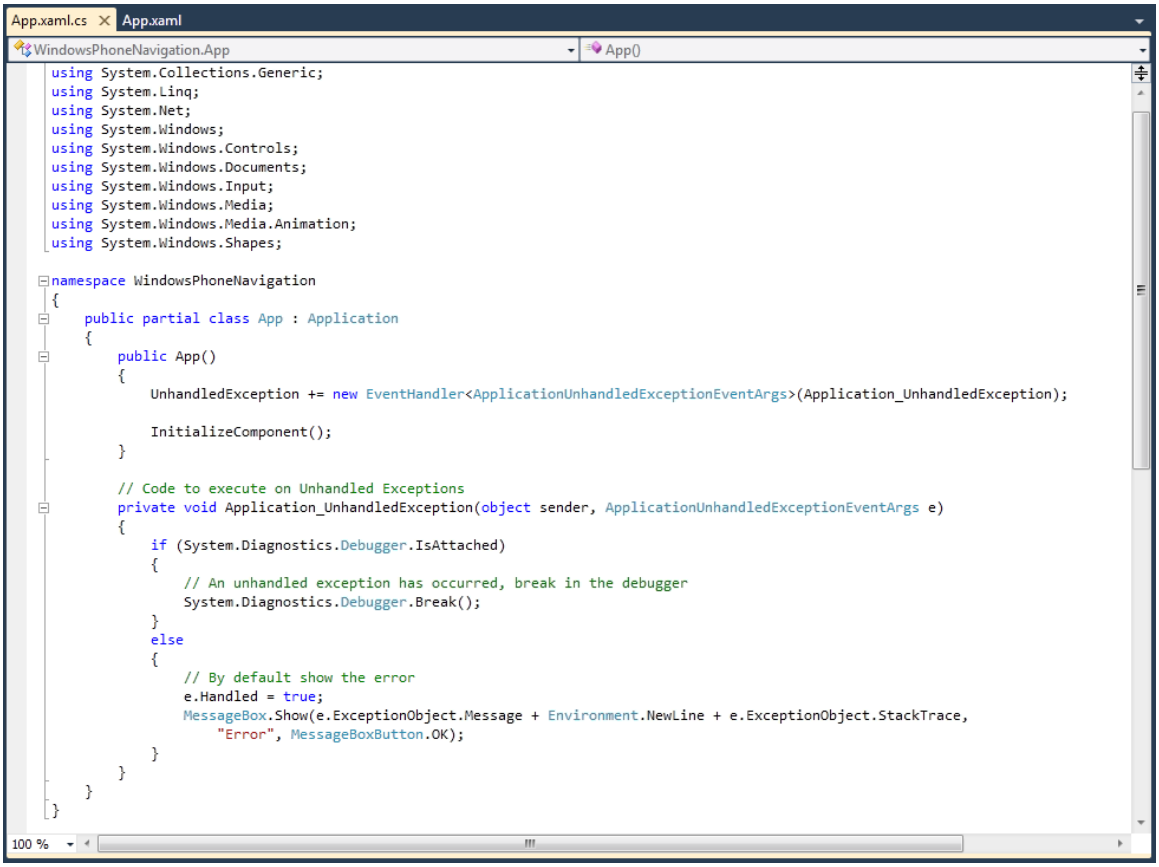


Figure 1

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.

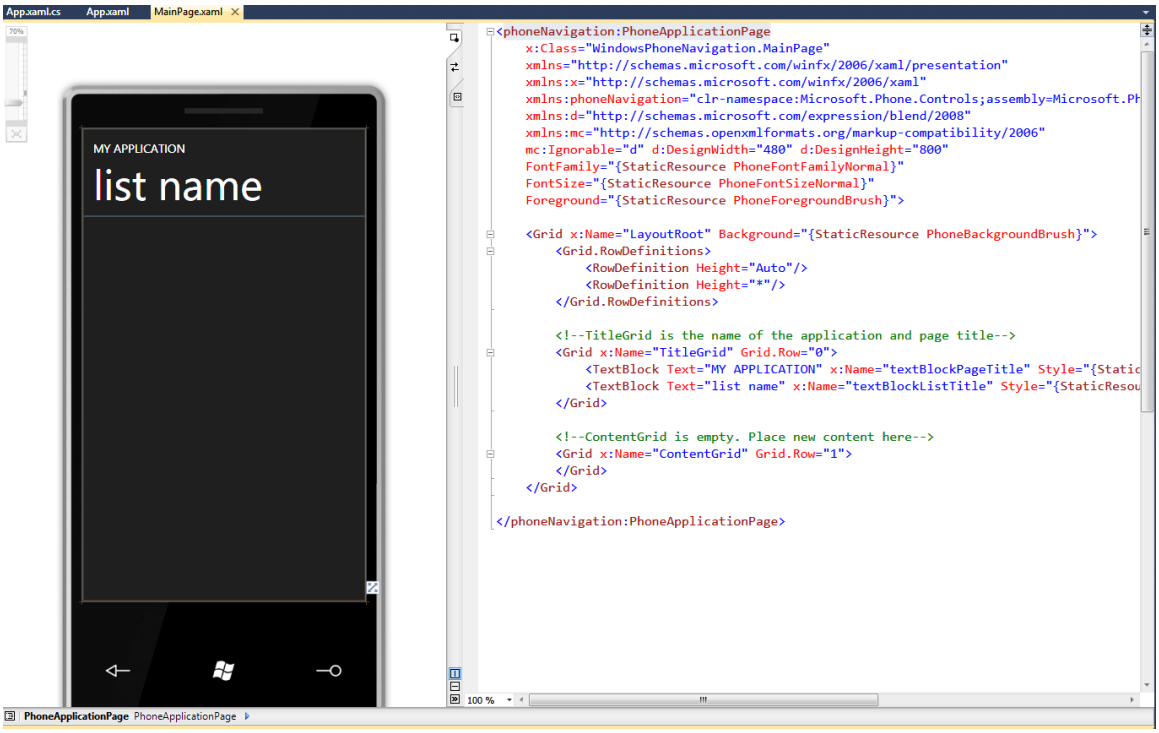


Figure 5

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.

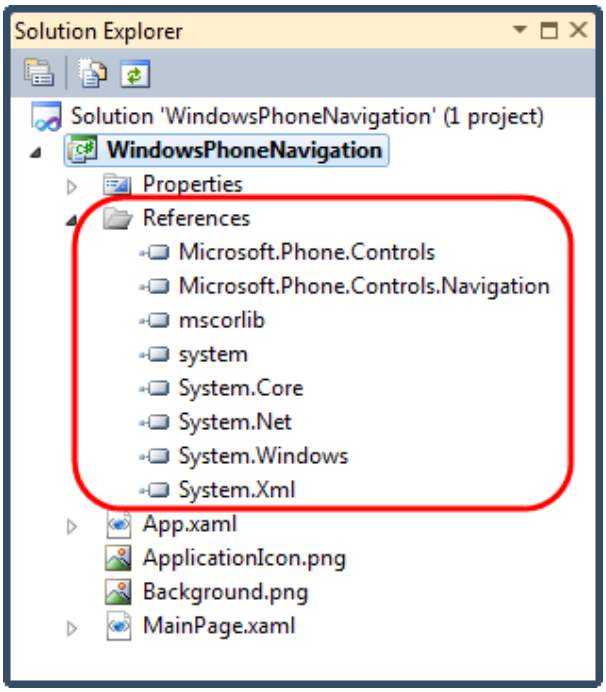


Figure 6

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.

XAML

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

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.

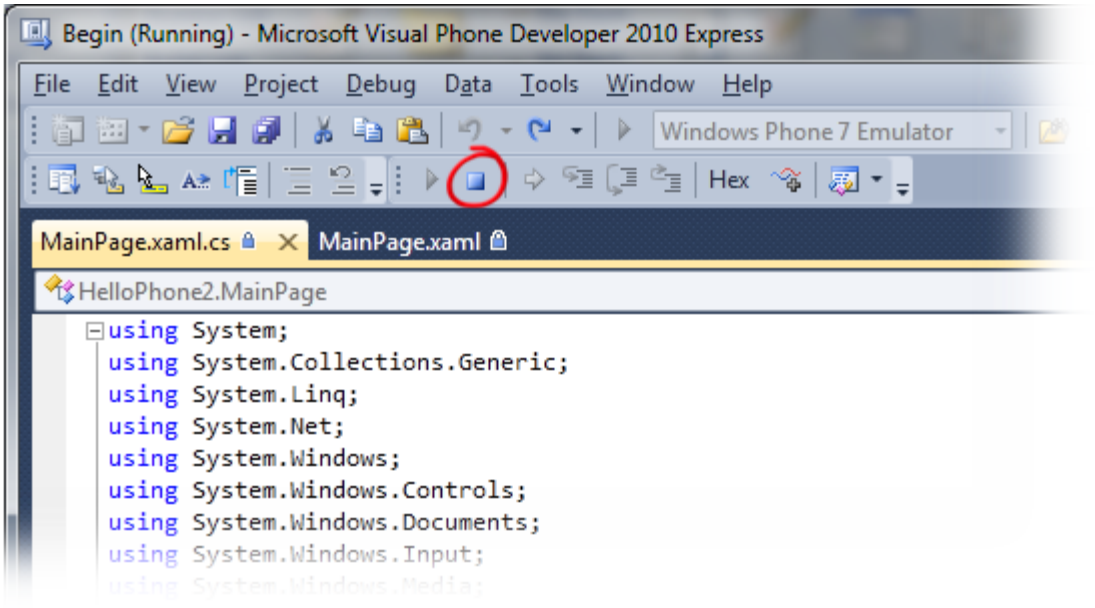


Figure 3

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**

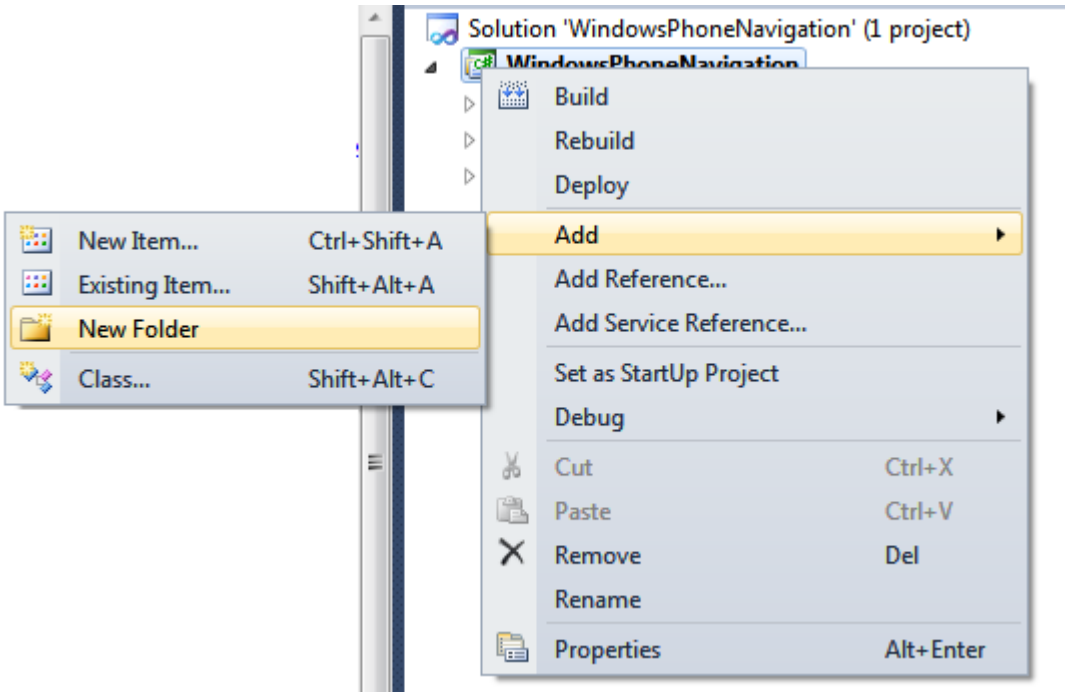


Figure 4

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 :

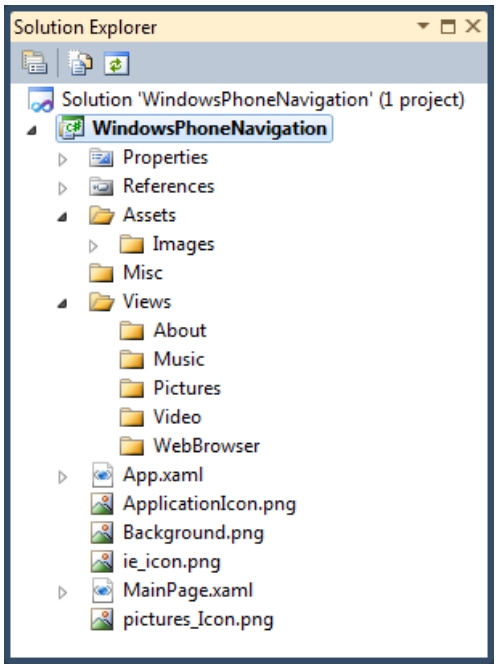


Figure 5

Structure du projet

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

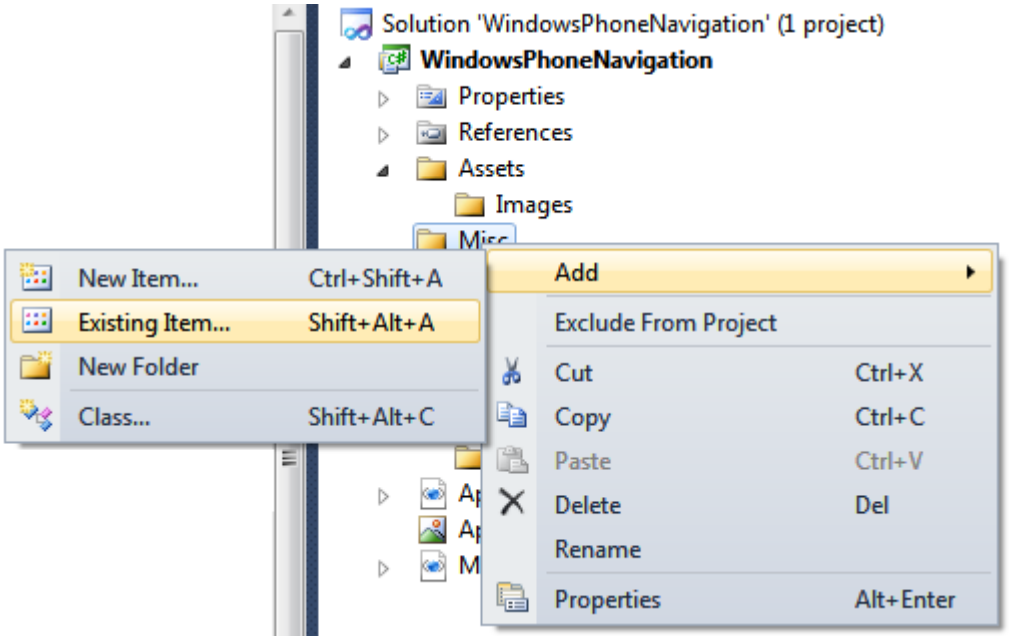


Figure 6

Ajout d'éléments existants aux dossiers du projet

Localisation de l'Asset	Nom de l'Asset	Dossier projet
Assets\Classes	Photo.cs Utils.cs	\Misc
Assets\Music	Music1.mp3 Music2.mid	\Assets
Assets\Video	Video1.wmv Video2.wmv	\Assets
Assets\Pictures	Toutes les images	\Assets\Images

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 ajoutés (à l'exception des fichiers portant l'extension « .cs ») en ressources Content :
    - Faites un clic-droit sur le fichier ressource
    - Sélectionnez **Properties.**
    - Sélectionnez **Content**.
    - Faites de même pour tous les fichiers de ressources autres que les fichiers .cs

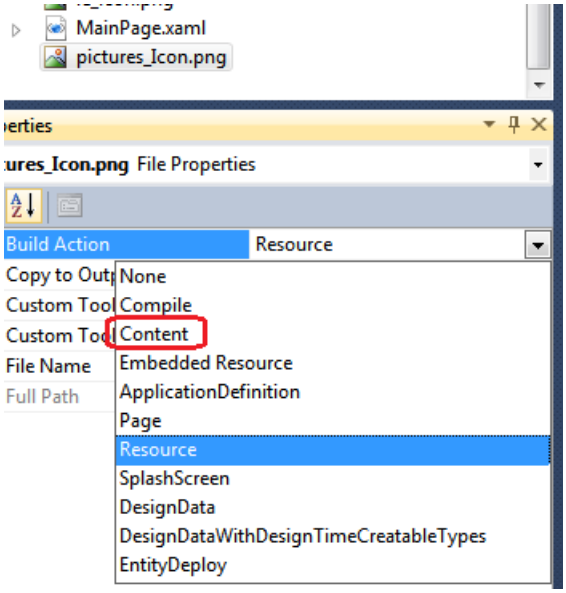


Figure 7

Modification du type de ressource

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

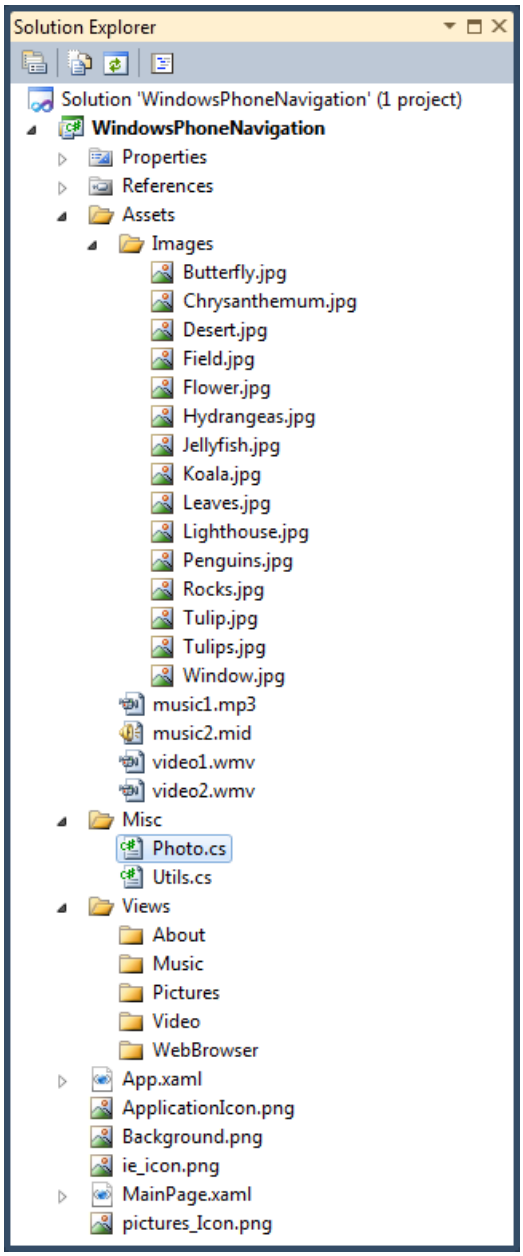


Figure 8

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**.

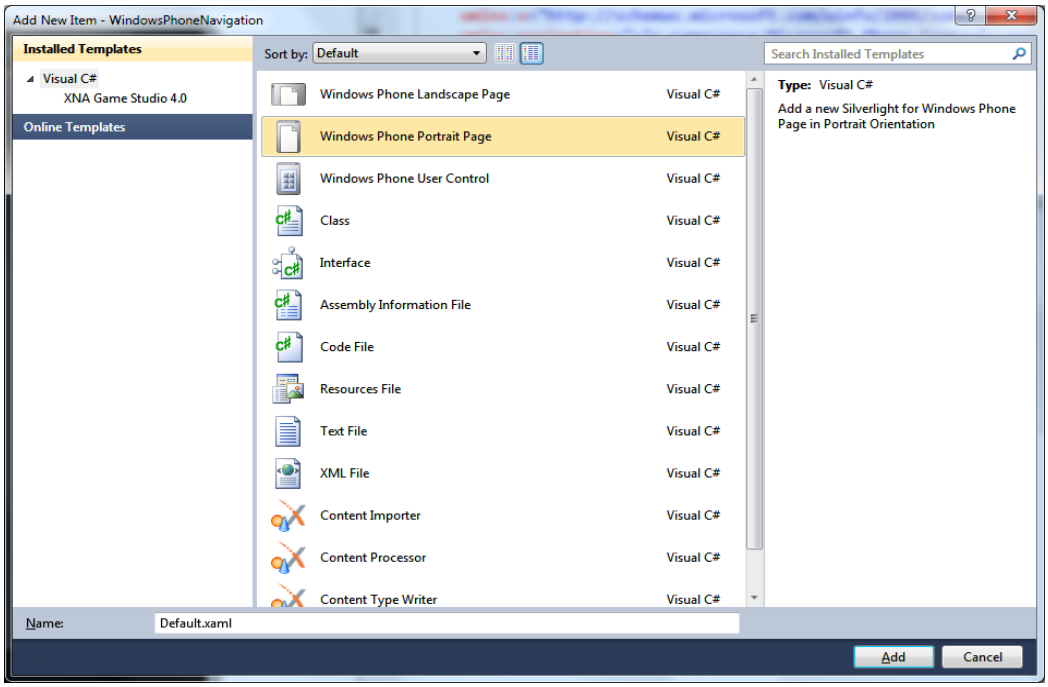


Figure 9

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.

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** :

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 :

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.Navig
ation"
```

> **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.

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 :

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 :

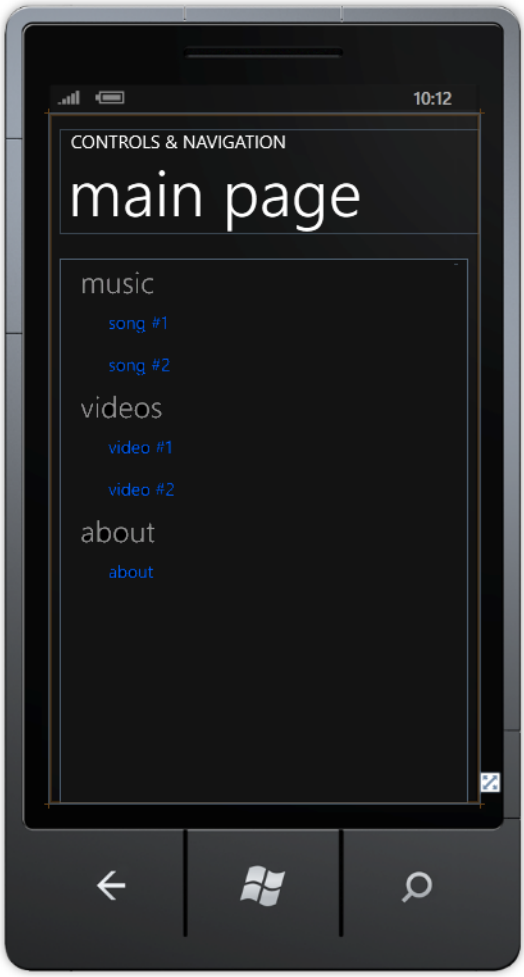


Figure 10

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.

Exercice 4 – 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

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.

12. Ouvrez la solution créée dans l'exercice précédent.
13. Afin d'ajouter une gestion automatique de l'orientation des pages, définissez l'attribut *SupportedOrientations* dans l'objet Page.
14. 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..

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

Figure 11

La page 'About' en orientation automatique

16. Arrêtez le débogage et retourner au code de l'application.
17. 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.
18. 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 :

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);
```

```
    }
};
```

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

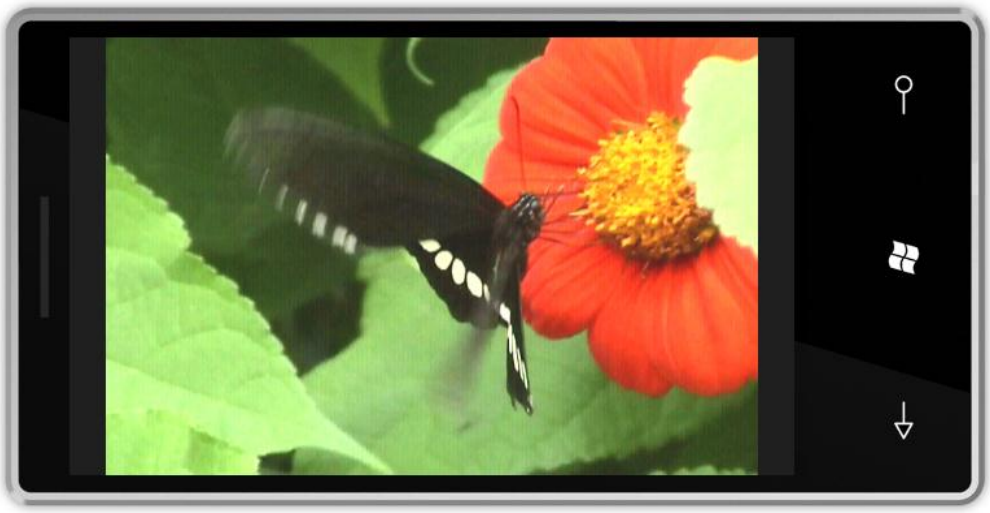


Figure 12

Affichage d'une vidéo en mode paysage

20. 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

```
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**.

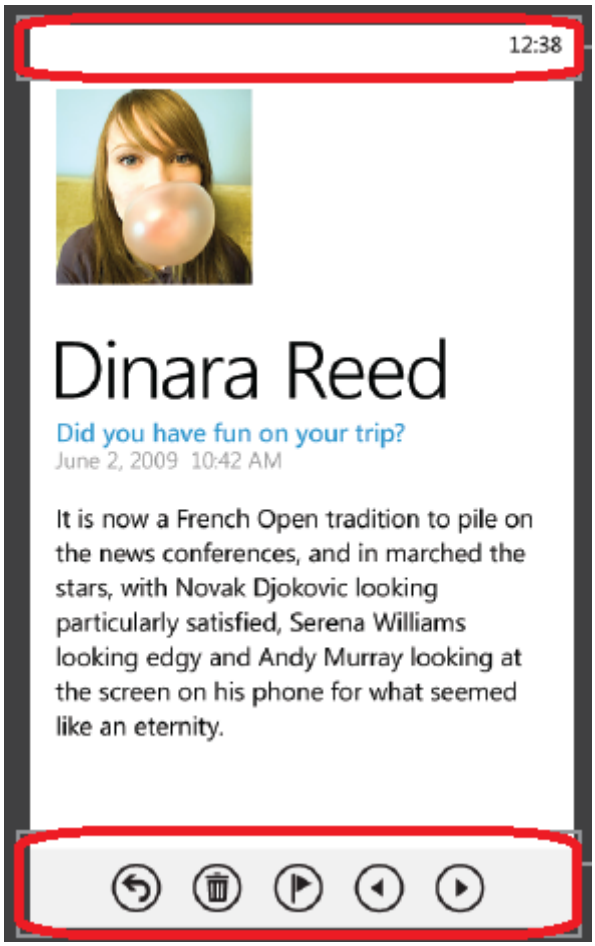


Figure 13

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

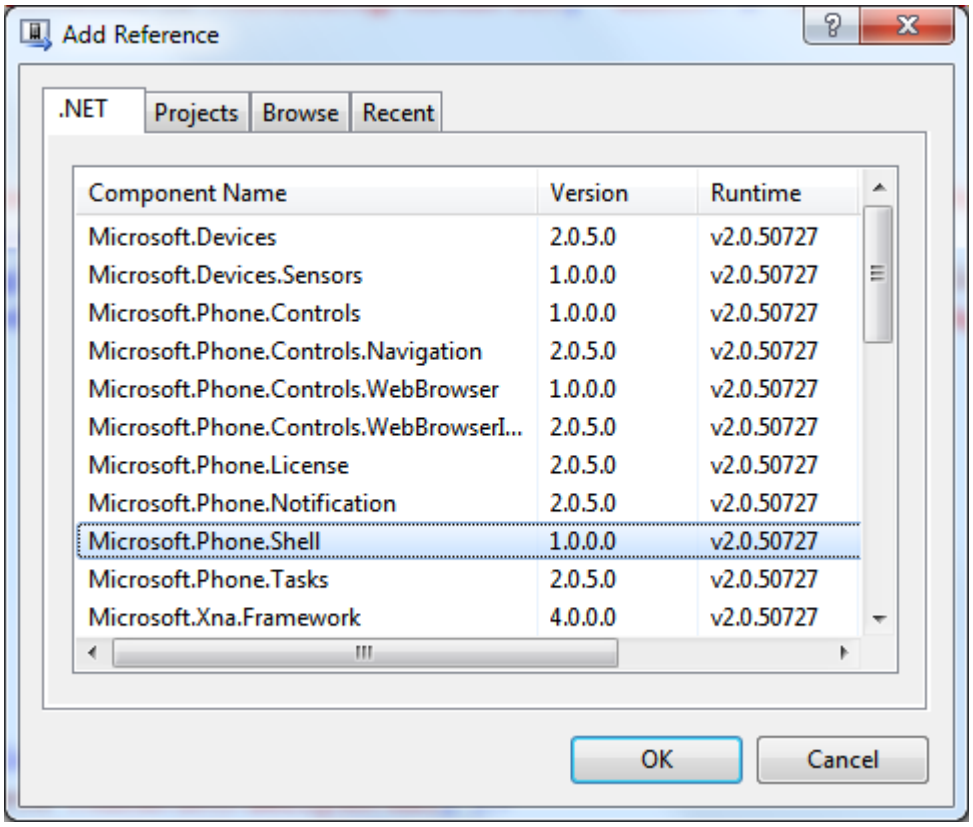


Figure 14

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 :

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

```
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*

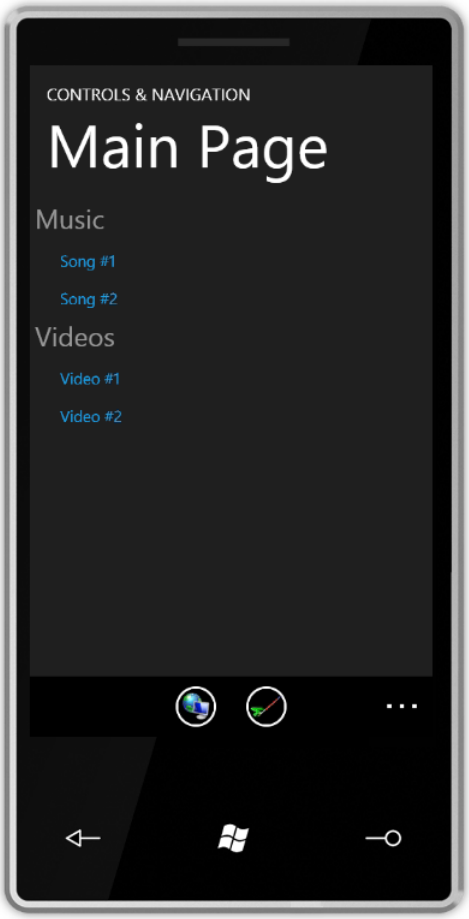


Figure 15

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 :

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 :**

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.

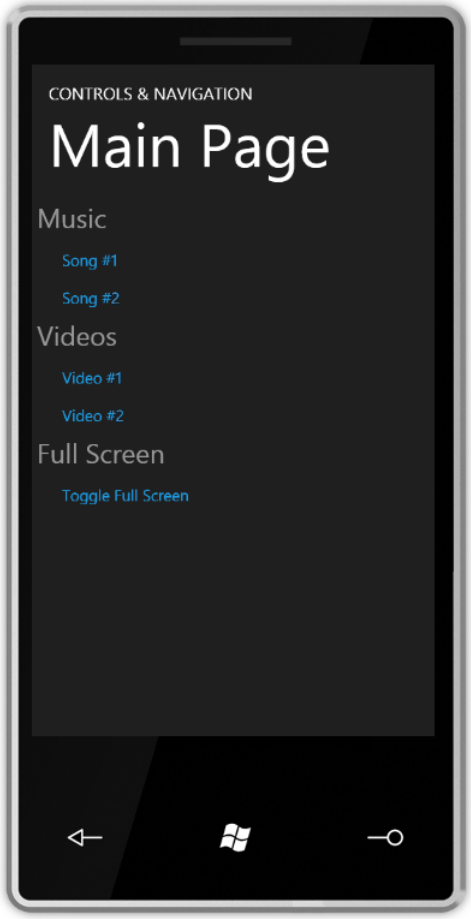


Figure 16
L'application en plein écran

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") :

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") :

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 :

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 :

```
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".

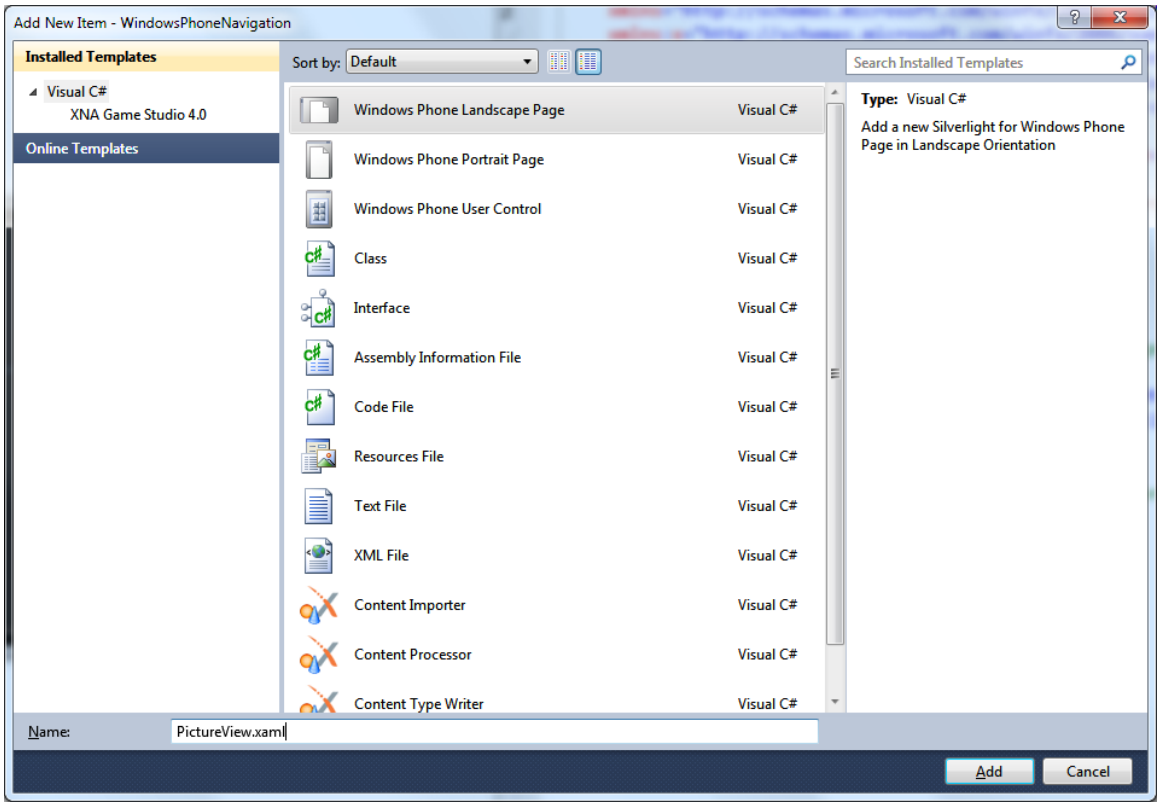


Figure 17

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 :

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 :

```
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 :

```
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.

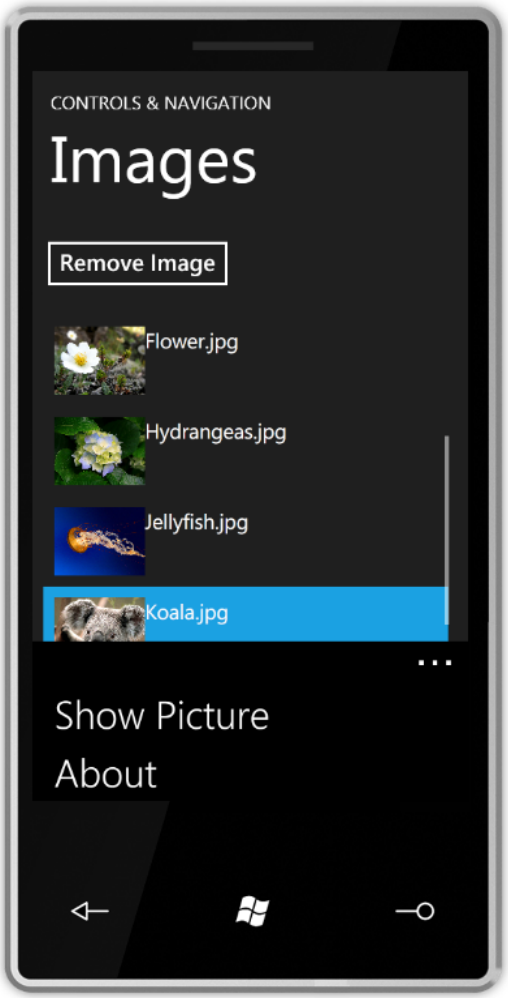


Figure 18

Application Bar avec les elements de menu

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

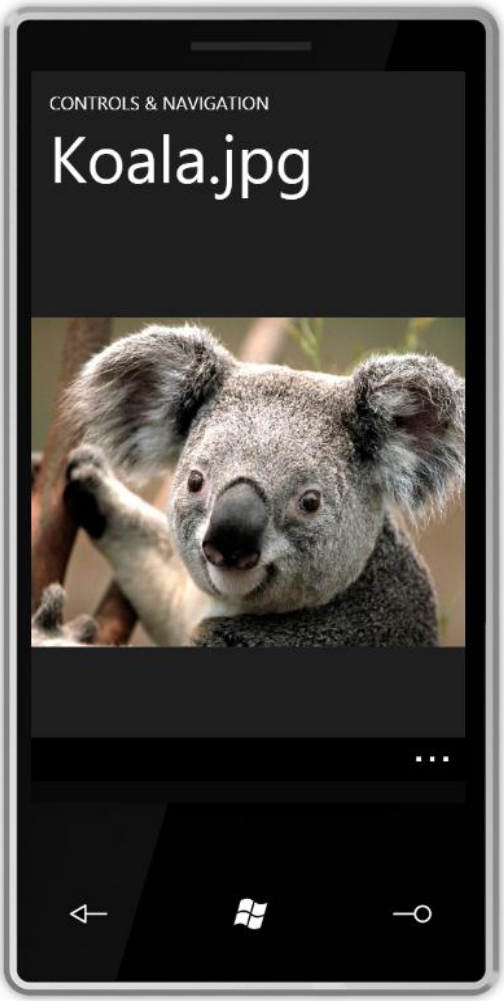


Figure 19

Page PictureView en orientation Portrait

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

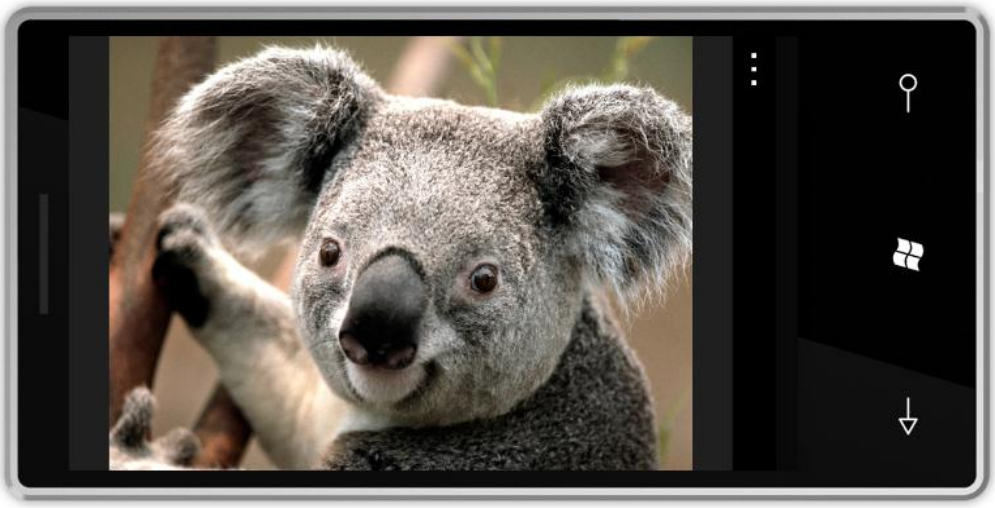


Figure 20

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 ».

Exercice 5 – Utilisation des tâches du téléphone

Les applications Windows Phone 7 fonctionnent dans un environnement d’exécution isolé. Le seul moyen pour elles de récupérer des informations depuis le système d’informations (contacts, adresses, photos) ou d’effectuer des actions sur applications natives (composition de numéro de téléphone ou d’email) est l’utilisation de tâches, pour les actions sans valeur de retour, et de « choosers » lorsqu’un retour est attendu.

Dans cet exercice, vous découvrirez les différetes tâches et « choosers » disponibles dans le SDK du téléphone.

1. Créez une nouvelle application Silverlight pour Windows Phone
2. Ouvrez le fichier MainPage.xaml et localisez la ligne suivante :

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

3. Sous cette ligne, ajoutez le code suivant, qui permet de déclarer un StackPanel contant plusieurs boutons qui nous serviront à lancer nos différentes tâches :

C#
```
<ScrollViewer>
    <StackPanel Orientation="Vertical"
    HorizontalAlignment="Center">
        <Button Content="EmailComposeTask"
    Click="OnEmailComposeTaskClick" />
        <Button Content="PhoneCallTask"
    Click="OnPhoneCallTaskClick" />
        <Button Content="PhotoChooserTask"
    Click="OnPhotoChooserTaskClick" />
       <Image x:Name="photo" Width="100" Height="100" Visibility="Collapsed"
        />
        <Button Content="AddressChooserTask"
    Click="OnAddressChooserTaskClick" />
        <Button Content="AddressChooserTask"
    Click="OnAddressChooserTaskClick" />
        <Button Content="BingMapDirectionsTask"
    Click="OnBingMapDirectionsTaskClick" />
        <Button Content="ConnectionSettingsTask"
    Click="OnConnectionSettingsTaskClick" />
```

```
        <Button Content="SaveContactTask"
    Click="OnSaveContactTaskClick" />
        <Button Content="SaveRingtoneTask"
    Click="OnSaveRingtoneTaskClick" />
    </StackPanel>
</ScrollViewer>
```

4. Ouvrez à présent le fichier MainPage.xaml.cs et ajoutez les méthodes suivantes, qui permettent au code de compiler mais qui, pour le moment, ne font rien. Ces méthodes seront appelées lorsque l'utilisateur cliquera sur un des boutons de l'interface graphique :

C#

```
private void OnEmailComposeTaskClick(object sender, RoutedEventArgs e)
{
}

private void OnPhoneCallTaskClick(object sender, RoutedEventArgs e)
{
}

private void OnPhotoChooserTaskClick(object sender, RoutedEventArgs e)
{
}

private void OnAddressChooserTaskClick(object sender, RoutedEventArgs e)
{
}

private void OnBingMapDirectionsTaskClick(object sender, RoutedEventArgs e)
{
}

private void OnConnectionSettingsTaskClick(object sender, RoutedEventArgs e
)
{
}

private void OnSaveContactTaskClick(object sender, RoutedEventArgs e)
{
}

private void OnSaveRingtoneTaskClick(object sender, RoutedEventArgs e)
{
}
```

5. Dès lors, si vous exécutez l'application, vous constaterez quelle se lance correctement et qu'il ne reste plus qu'à appeler le code correspondant aux différentes tâches :

La tâche EmailComposeTask

Cette première tâche permet d'ouvrir automatiquement la fenêtre de composition d'un message électronique. Ils'agit bien d'une tâche et non d'un « chooser », il n'y a pas de valeurs de retour disponible.

1. Remplacer le code de la méthode OnEmailComposeTaskClick par le code ci-dessous, qui ouvrira votre client mail avec un nouveau message précomplété pour les champs **Subject** et **Body**.

C#

```
private void OnEmailComposeTaskClick(object sender, RoutedEventArgs e)
{
    var emailComposeTask = new EmailComposeTask();

    emailComposeTask.Subject = "Une info à partager !";
    emailComposeTask.Body = "La description de mon info.";

    emailComposeTask.Show();
}
```

2. Pour que cela fonctionne, il est nécessaire d'ajouter la directive **using** correspondant à l'espace de nom dans lequel se trouvent les différentes tâches :

C#

```
using Microsoft.Phone.Tasks;
```

3. Attention, cette tâche ne fonctionne pas sur l'émulateur, vous devez la tester sur un mobile.

La tâche PhoneCallTask

Cette tâche va vous permettre de composer directemetun numéro de téléphone à la place de l'utilisateur, celui-ci n'ayant plus qu'à appuyer sur le bouton « Appeler » pour passer son appel. Encore une fois, il s'agit d'une tâche et non d'un « chooser », il n'y a pas de valeur de retour disponible.

1. Remplacer le code de la méthode OnPhoneCallTaskClick par le code ci-dessous qui initialise la tâche avec un numéro de téléphone et le nom à afficher dans la fenêtre d'appel.

C#

```
private void OnPhoneCallTaskClick(object sender, RoutedEventArgs e)
{
    var phoneCallTask = new PhoneCallTask();
    phoneCallTask.PhoneNumber = "0123456789";
    phoneCallTask.DisplayName = "Mr Smith";
    phoneCallTask.Show();
}
```

2. A l'exécution, lorsque vous pressez le bouton **PhoneCallTask**, une boite de dialogue apparaît vous proposant d'appeler le numéro défini précédemment dans le code.

La tâche PhotoChooserTask

Le chooser PhotoChooserTask va vous permettre de demander à l'utilisateur de choisir une photo depuis la gallerie d'image et éventuellement lui laisser la possibilité de prendre une nouvelle image

depuis la caméra du téléphone. Il est également possible de définir la taille maximale de l'image à récupérer pour que le téléphone demande automatiquement à l'utilisateur de cropper cette image.

1. Ajoutez une directive using pointant vers l'espace de nom **System.Windows.Media.Imaging**.

2. Remplacez le code de la méthode **OnPhotoChooserTaskClick** avec l'extrait de code ci-dessous. Notez que la syntaxe est différente de celles des deux précédentes tâches : ici, une callback est définie sur l'évènement **Completed**. Cette callback recevra comme arguments des informations sur l'image sélectionnée par l'utilisateur (l'image en elle-même, le nom du fichier, etc.). Le stream reçu est ensuite passé à un objet de type **BitmapImage** qui sera utilisé dans le contrôle **Image** précédemment placé dans le **StackPanel**.

C#

```
private void OnPhotoChooserTaskClick(object sender, RoutedEventArgs e)
{
    var photoChooserTask = new PhotoChooserTask();

    photoChooserTask.Completed += (s, args) =>
        {
            var image = new BitmapImage();
            image.SetSource(args.ChosenPhoto);
            photo.Source = image;
            photo.Visibility = System.Windows.Visibility.Visible;
        };

    photoChooserTask.Show();
}
```

3. Exécutez l'application. En cliquant sur le bouton, la gallerie d'image s'ouvre. Après avoir choisi une image, vous revenez dans l'application et l'image sélectionnée apprait dans le **StackPanel**.

La tâche AddressChooserTask

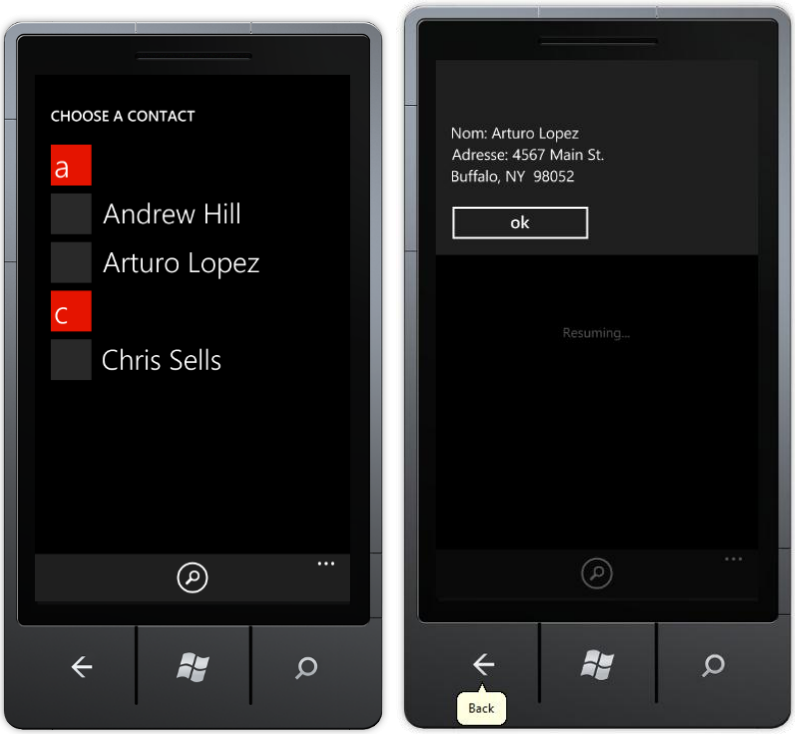
Cette tâche permet, comme vous pouvez vous en doutez grâce à son nom, de sélectionner l'adresse d'un contact issu du téléphone. Son utilisation est très similaire à celle des tâches PhoneNumberChooserTask ou EmailAddressChooser qui existent déjà dans Windows Phone.

1. Remplacer le code de la méthode OnAddressChooserTaskClick par le code ci-dessous, qui permet, sur le clic du bouton, d'afficher le contrôle permettant de lister les contacts puis, à la validation, d'afficher des informations sur le contact sélectionné :

C#

```
private void OnAddressChooserTaskClick(object sender, RoutedEventArgs e) {
    var addressChooserTask = new AddressChooserTask();
    addressChooserTask.Completed += (s, args) => {
        if (args.TaskResult == TaskResult.OK) {
            MessageBox.Show(string.Format("Nom: {0}{1}Adresse: {2}", args.D
isplayName, Environment.NewLine, args.Address));
        }
    };
    addressChooserTask.Show();
}
```

2. Si vous exécutez l'application et que vous cliquez sur le premier bouton, vous constaterez que l'application affiche bien la liste des contacts de votre téléphone (si vous êtes sur l'émulateur, vous pouvez également voir que certains contacts fictifs sont également présents, permettant de simplifier les scénarios de test). A la sélection, le nom et l'adresse du contact sélectionné sont affichés :



Les tâches BingMapsDirectionsTask et BingMapTask

Cette tâche peut être utilisée dans les applications qui proposent de la géolocalisation et permet au développeur d'afficher un itinéraire entre un point de départ (généralement la position de l'utilisateur) et un point d'arrivée.

1. Modifier le code de la méthode OnBingMapDirectionsTaskClick pour le faire ressembler à celui-ci :

C#

```
var bingMapsDirectionTask = new BingMapsDirectionsTask();
bingMapsDirectionTask.Start = new LabeledMapLocation("Maison", new GeoCoord
inate(48.4817, 2.0715));
bingMapsDirectionTask.End = new LabeledMapLocation("Travail", new GeoCoordi
nate(48.4817, 2.0815));
bingMapsDirectionTask.Show();
```

2. Attention, pour que ce code fonctionne, il est nécessaire d'ajouter une référence à **Sytem.Device.dll** !
3. A présent, si vous lancez l'application et que vous cliquez sur le 2ème bouton, vous pourrez constater que l'itinéraire est correctement affiché (et vous avez même la possibilité d'indiquer si vous êtes à pieds ou en voiture) :

La tâche BingMapsTask est légèrement similaire à celle-ci dans le sens où son objectif est de vous permettre d'afficher l'application BindMaps, sur des coordonnées précises. Vous pouvez également lui spécifier un niveau de zoom, permettant à l'utilisateur d'être plus ou moins prêt de l'endroit rechercher.

La tâche ConnectionSettingTask

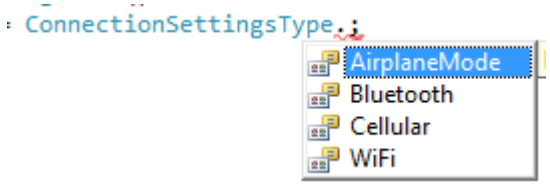
Cette tâche fait suite à une remarque de la part des utilisateurs qui se plaignait de devoir quitter leur application favori pour être obliger de changer les réglages de connexion du téléphone (coupure ou activation du Wifi, etc.). Il est maintenant possible de lancer ces différents écrans directement depuis l'application, sans avoir besoin de la quitter pour la relancer.

1. Localiser la méthode OnConnectionSettingsTaskClick et remplacer son contenu par celui-ci :

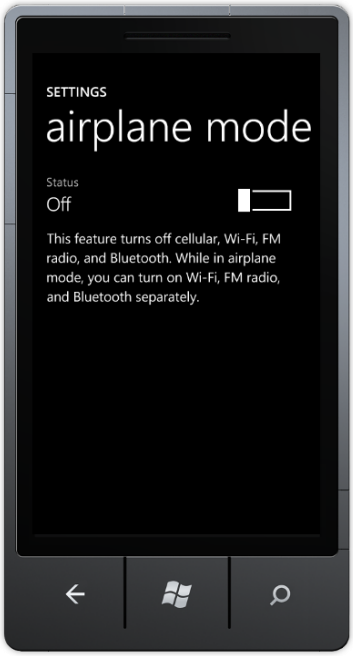
C#

```
var connectionSettingsTask = new ConnectionSettingsTask();
connectionSettingsTask.ConnectionSettingsType = ConnectionSettingsType.Airp
laneMode;
connectionSettingsTask.Show();
```

2. La propriété ConnectionSettingsType est utilisée pour indiquer quel paramètre de connexion votre application gérer. La valeur de cette propriété est une énumération dont les choix possible sont visibles ci-dessous :



3. Si vous lancez l'application et que vous cliquez sur le bouton « ConnectionSettingsTask », vous constaterez que l'écran d'activation du mode avion s'affiche correctement. Dès lors, vous pouvez modifier les paramètres et un simple clic sur le bouton « Back » vous permet de retourner dans l'application !



Les tâches SaveContactTask et SaveRingtoneTask

La classe SaveContactTask permet au développeur de proposer des applications qui offrent la possibilité d'enregistrer, directement dans le téléphone, un nouveau contact sans que l'utilisateur soit obligé de quitter l'application.

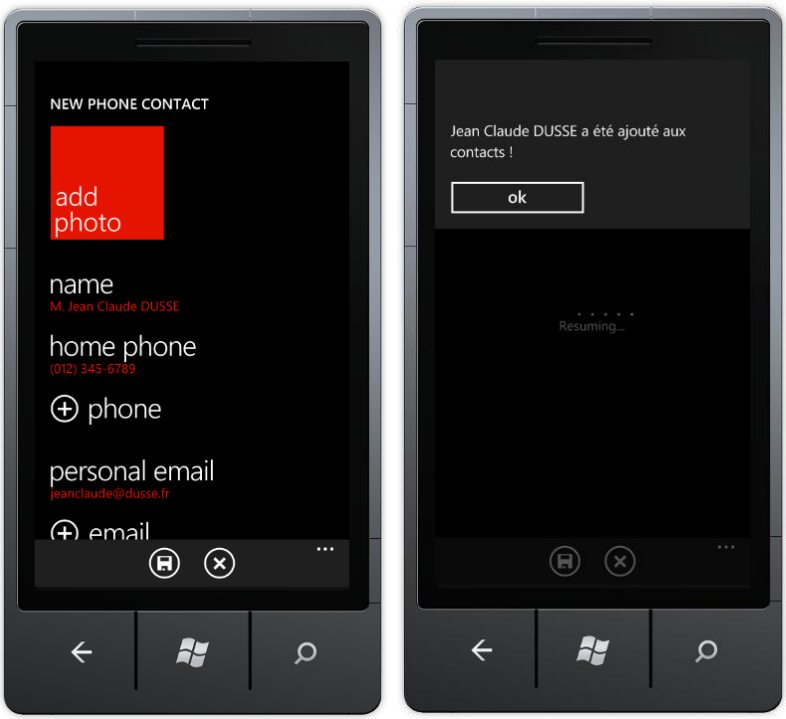
Tout un ensemble de propriétés (FirstName, LastName, Tile, WorkEmail, etc.) peuvent être utilisées pour remplir les informations du contact.

1. Modifier le code de la méthode OnSaveContactTaskClick pour le faire ressembler au code ci-dessous :

C#

```
var saveContactTask = new SaveContactTask();
saveContactTask.Title = "M.";
saveContactTask.LastName = "DUSSE";
saveContactTask.FirstName = "Jean Claude";
saveContactTask.HomePhone = "0123456789";
saveContactTask.JobTitle = "Dragueur";
saveContactTask.PersonalEmail = "jeanclaude@dusse.fr";
saveContactTask.Website = "http://www.jeanclaudedusse.net";

saveContactTask.Completed += (s, ars) => {
    if (ars.TaskResult == TaskResult.OK)     {
         MessageBox.Show(string.Concat(saveContactTask.FirstName, " ", save
ContactTask.LastName, " a été ajouté aux contacts !"));
     }
};

saveContactTask.Show();
```

2. Ce code permet de créer un nouveau contact et d'afficher la fenêtre correspondante à l'utilisateur. Attention, il est important de noter qu'il n'est pas possible de créer réellement le contact : on se contente de remplir les informations mais c'est l'utilisateur qui, au final, décide de s'il enregistrera ou non le contact



SaveRingtoneTask est une tâche qui peut permettre de contourner une des limitations actuelles de Windows Phone en ce qui concerne l'utilisation de sonneries personnalisées. En effet, cette tâche va permettre de proposer un fichier (MP3 ou WMA) comme sonnerie utilisable sur le téléphone.

Pour que cela fonctionne, il faut noter les limitations concernant les fichiers de son :

- Ils doivent être au format MP3 ou WMA
- Ils doivent durer moins de 40 secondes
- Ils doivent peser moins de 1 Mo
- Ils ne doivent pas être protégés par des DRM

Autre point important, chacun des fichiers son doit obligatoirement être contenu dans le XAP en tant que ressource ou être stockés dans l'Isolated Storage (on peut ainsi imaginer télécharger un fichier depuis Internet, le stocker dans l'Isolated Storage et l'utiliser comme sonnerie).

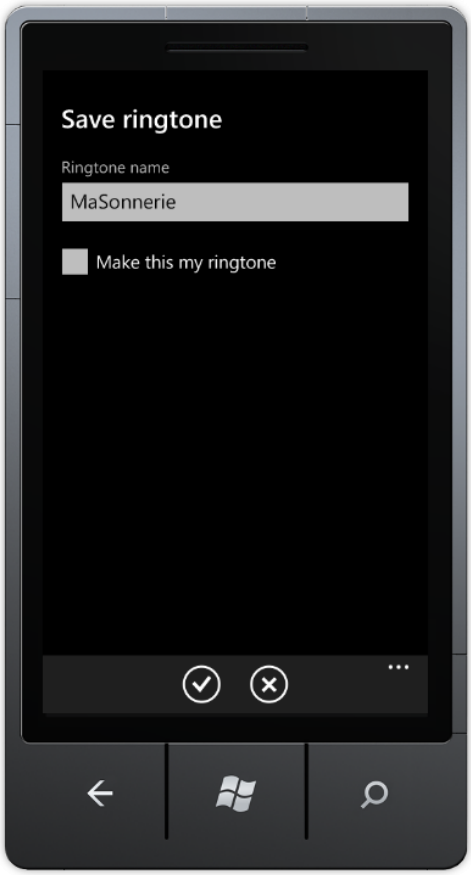
1. Modifiez le corps de la méthode OnSaveRingtoneTaskClick pour le faire ressembler au code ci-dessous :

C#

```
string fileName = "MaSonnerie.mp3";

var wc = new WebClient();
wc.OpenReadCompleted += (s, args) => {
    using (var store = IsolatedStorageFile.GetUserStoreForApplication())
    {
        if (store.FileExists(fileName))
        {
            store.DeleteFile(fileName);
        }
```

```
        using (var fs = new IsolatedStorageFileStream(fileName, FileMode.Cre
ate, store)) {
            var byteInStream = new byte[args.Result.Length];
            args.Result.Read(byteInStream, 0, (int)byteInStream.Length);
            fs.Write(byteInStream, 0, byteInStream.Length);
            fs.Flush();
        }

        var saveRingToneTask = new SaveRingtoneTask();
        saveRingToneTask.DisplayName = "MaSonnerie";
        saveRingToneTask.Source = new Uri(string.Format("isostore:/{0}", fil
eName));
        saveRingToneTask.Completed += (source, a) => {
            if (a.TaskResult == TaskResult.OK)
            {
                MessageBox.Show("Sonnerie sauvegardée !");
            }
        };
        saveRingToneTask.Show();
    }
};
wc.OpenReadAsync(new Uri("http://nekomimimusic.mduo13.com/music/ElCazadorOST
Demo.mp3"));
```

2. Ce code permet de télécharger un fichier MP3 depuis Internet, de le stocker dans l'Isolated Storage puis, une fois le téléchargement et l'enregistrement effectué, de définir ce fichier comme étant une sonnerie du téléphone (avec, en prime, la possibilité d'indiquer qu'il s'agit de la sonnerie par défaut)



3. Attention, bien que fonctionnel, le code ci-dessus ne prend pas en compte les erreurs qui peuvent survenir lors du téléchargement du fichier, l'annulation de l'enregistrement de la sonnerie ou bien, tout simplement, la mise en place d'une barre de progression permettant aux utilisateurs de savoir ce que fait l'application (téléchargement du fichier, enregistrement, etc.).

Les autres tâches

D'autres tâches existent et peuvent être utilisées avec Windows Phone. Ainsi, on peut noter la présence des tâches suivantes :

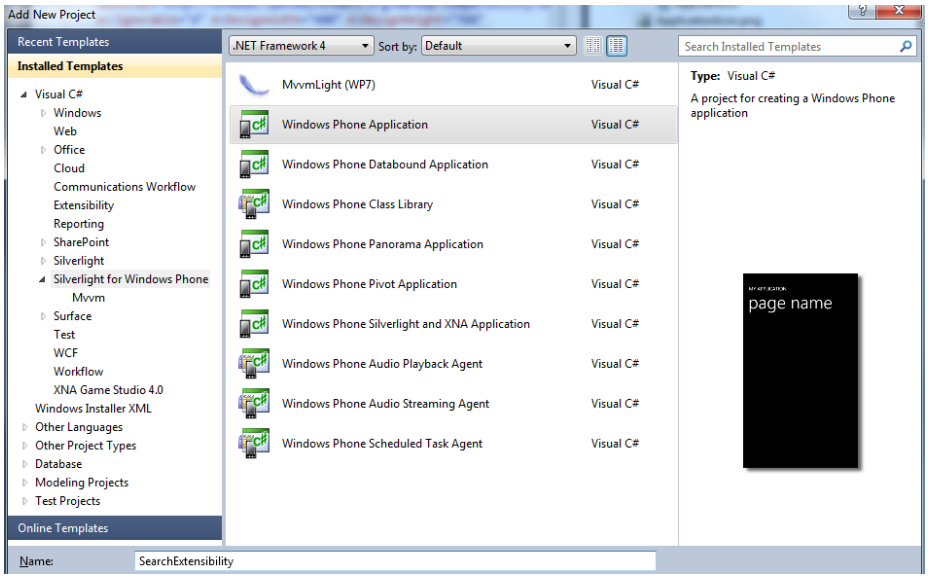
- **GameInviteTask**, qui permet d'inviter un contact pour jouer à plusieurs
- **CameraCaptureTask**, qui permet de récupérer une image depuis la caméra du téléphone
- **EmailAddressChooserTask**, qui permet de récupérer une adresse email parmis les contacts de l'utilisateur
- **MarketplaceDetailTask**, qui permet de voir le détail d'une application sur le marketplace, soit pour l'application courante ou alors pour une autre application via son ID
- **MarkeplaceHubTask**, qui permet d'ouvrir le hub Marketplace
- **MarketplaceReviewTask**, qui permet d'afficher les commentaires d'une application, soit pour l'application courante ou alors pour une autre application via son ID
- **MarketplaceSearchTask**, qui permet de lancer une recherche sur le Marketplace et d'afficher les résultats
- **SmsComposeTask**, qui permet d'ouvrir la fenêtre de composition de SMS et de la pré remplir
- **WebBrowserTask**, qui permet d'ouvrir le navigateur du téléphone vers une URL donnée
- **ShareStatusTask**, qui offre la possibilité aux développeurs de partager le statut des utilisateurs sur les réseaux sociaux directement depuis l'application !
- **ShareLinkTask** qui, lorsqu'elle est invoquée, permet de partager un lien sur les réseaux sociaux (là encore directement depuis l'application, sans avoir besoin de lancer une application tierce).

Exercice 6 – Intégration aux hubs du téléphone

Une des nouvelles fonctionnalités disponible dans Windows Phone 7.5 porte le nom de code « App Connect ». Il s'agit de la possibilité d'intégrer le contenu des applications au sein des résultats de recherche, ainsi que des hubs Musiques, Photos et Vidéos.

Ainsi, lors d'une recherche sur Bing, il est par exemple possible de prendre une photo et d'appeler l'application de manipulation de photos que vous avez développé, directement depuis la page de résultats de recherche, sans avoir besoin de revenir sur la page d'accueil. Grâce à cette fonctionnalité, les applications apparaissent lorsqu'elles sont vraiment utiles, en fonction du besoin, du contexte, etc.

1. Créez un nouveau projet Silverlight pour Windows Phone que vous nommerez « SearchExtensibility » :



2. Ouvrez le fichier **WMAppManifest.xml** qui est dans le répertoire « *Properties* » et ajoutez le code suivant, avant la balise « </App> » :

C#

```
<Extensions>
<Extension ExtensionName="Bing_Products_Video_Games"
                ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFEA5661}"
                TaskID="_default"
                ExtraFile="Extensions\\Extras.xml" />
</Extensions>
```

3. Chaque élément « Extension » utilise les 4 attributs suivants pour décrire l'extension que vous êtes en train d'écrire :
    a. ExtensionName : qui permet d'indiquer le nom de l'extension. Pour rappel, une extension est une catégorie spécifique de résultat de recherche.
    b. ConsumerID : qui reconnait l'extension comme une extension de la recherche. Toutes les extensions de recherche nécessitent la même valeur : 5B04B775-356B-4AA0-AAF8-6491FFEA5661
    c. TaskID : qui correspond au nom de la tâche pas défaut qui est indiqué dans l'élément Task. Généralement, cette valeur (si vous ne l'avez pas changé) est « _default ».
    d. ExtraFile : qui pointe vers l'emplacement du fichier Extras.xml. Ce fichier doit être dans un répertoire nommé Extensions et doit s'appeler Extras.xml.
4. A présent, créez un répertoire nommé Extensions et, à l'intérieur, créez un fichier Extras.xml. Remplacez le contenu de ce fichier par le code ci-dessous, qui permet d'indiquer les labels qui apparaitront dans les résultats de recherche :

C#

```
<?xml version="1.0" encoding="utf-8" ?>
<ExtrasInfo>
  <AppTitle>
    <default>Mes Produits</default>
  </AppTitle>
```

```
  <Consumer ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFEA5661}">
    <ExtensionInfo>
      <Extensions>
 <ExtensionName>Bing_Products_Video_Games</ExtensionName>
      </Extensions>
      <CaptionString>
        <default>Recherche dans Mes Produits</default>
      </CaptionString>
    </ExtensionInfo>
  </Consumer>
</ExtrasInfo>
```

5. Lorsqu'un utilisateur clique sur un résultat de recherche pour arriver directement dans votre application, « App Connect » génère un lien particulier (« Deep Link ») que votre application est en mesure d'intercepter pour rediriger la demande vers la bonne page. Pour cela, dans le fichier App.xaml, ajoutez la définition de namespace XML suivante :

C#

```
xmlns:nav="clr-
namespace:System.Windows.Navigation;assembly=Microsoft.Phone"
```

6. Puis, dans les ressources de la page, déclarez un UriMapper, qui servira à faire la redirection d'un utilisateur, lorsqu'il provient d'un résultat de recherche, vers la page principale :

C#

```
<nav:UriMapper x:Key="UriMapper">
    <nav:UriMapper.UriMappings>
        <nav:UriMapping Uri="/SearchExtras" MappedUri="/MainPage.xaml"/>
    </nav:UriMapper.UriMappings>
</nav:UriMapper>
```

7. Pour finir, il est nécessaire d'associer cet UriMapper à notre application (en effet, pour l'instant, il n'a simplement qu'été déclaré dans les ressources). Ouvrez le fichier App.xaml.cs et, dans le constructeur, ajoutez la ligne suivante :

C#

```
RootFrame.UriMapper = Resources["UriMapper"] as UriMapper;
```

8. A présent, il est nécessaire d'écrire le code qui récupérer l'élément que l'utilisateur a sélectionné dans les résultats de recherche. Pour cela, ouvrez le fichier MainPage.xaml.cs et sous l'appel à InitializeComponents, ajoutez cette ligne de code, qui est utilisée pour s'abonner à l'évènement Loaded de la page :

C#

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

9. Lorsque cet évènement survient, nous allons regarder si des paramètres sont passés dans l'URL et, le cas échéant, nous allons effectuer notre traitement (interrogation d'un Web Service, affichage de l'information, etc.). Pour cela, ajoutez le code suivante :

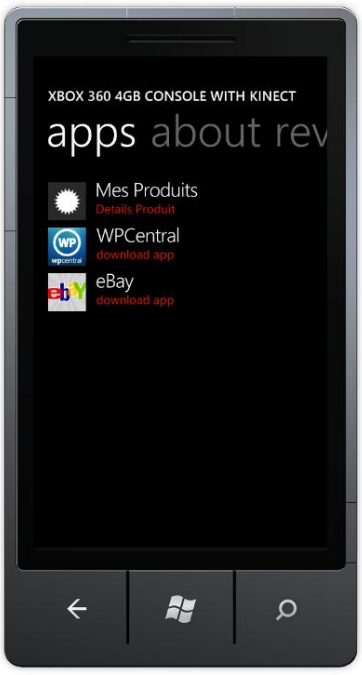
C#

```
private void MainPage_Loaded(object sender, RoutedEventArgs e) {
    foreach (string strKey in this.NavigationContext.QueryString.Keys) {
        if (strKey == "ProductName") {
           string productName = this.NavigationContext.QueryString[strKey];

            MessageBox.Show(productName);
         }
    }
}
```

10. Il ne reste plus qu'à tester l'application. Appuyez sur la touche F5 et, une fois que la page est chargée, appuyez sur le bouton de recherche pour faire apparaitre Bing :

11. Saisissez alors un mot qui fait partie de la catégorie ciblée. Pour rappel, dans les fichiers WMAppManifest.xml et Extras.xml, nous avons indiqué la catégorie « Bing_Products_Video_Games », il est donc nécessaire de faire une recherche en rapport avec cette catégorie. Ainsi, saisissez le mot « xbox kinect » et attendez de voir les résultats s'afficher :

12. Cliquez sur le résultat de recherche qui apparait sous la catégorie « products » et, dans la page qui apparait, faîtes défiler les PivotItems jusqu'à l'élément nommé « Apps » :



13. Vous constatez que notre application, telle que définie dans les fichiers XML, apparait bien. De plus, si l'on clique dessus, elle se charge et, comme l'utilisateur arrive depuis une page de résultats de recherche, le code écrit dans MainPage.xaml.cs se déclenche et on affiche, dans une MessageBox, le nom du produit sélectionné :

Résumé

Dans cet atelier, vous avez découvert les capacités de multi threading du téléphone. Vous avez pu comprendre le cycle de vie d'une application et les différents états qu'elle peut traverser. Vous avez également pu appréhender le système de navigation disponible sur Windows Phone. Enfin, vous avez découverts comment intéragir avec le système d'exploitation, soit en utilisant ses services, soit en utilisant ses tâches ou alors en s'intégrant dans ses hubs.