Hands-On Lab

Téléchargement de données et persistance

Version: 1.0.0

Dernière mise à jour : 1/10/2012

Sommaire

Vue d'ensemble

Cet atelier va vous présenter les techniques disponibles sur Windows Phone 7 pour télécharger des données et gérer ces données via différentes méthodes de stockage.

Objectifs

A la fin de cet atelier, vous aurez appris :

- Télécharger des données lorsque votre application est executée via le WebClient ou le couple WebRequest/WebRespons ;
- Faire communiquer des téléphones en utilisant les Sockets ;
- Télécharger des données en tâche de fond grâce au Background Transfer Service ;
- Manipuler des dossiers et des fichiers dans l'isolated storage ;
- Utiliser une base de données SQL CE.

Pré requis

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

- Microsoft Visual Studio 2010 Express pour Windows Phone 7
- SQL Server Management Studio Express

Exercices

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

1. Utilisation du Web Client
2. WebRequest et WebReponse
3. Les Sockets
4. Téléchargement en tâche de fond
5. Découverte de l'isolated storage
6. Utilisation d'une base de données locale
7. Utilisation d'une base de données SQL CE existante dans une application Windows Phone 7

Temps estimé pour compléter cet atelier: **180 minutes**.

Atelier : Téléchargement de données et persistance

Exercice 1 – Utilisation du Web Client

Dans cet exercice, allez découvrir une première approche pour télécharger des données depuis une application Silverlight pour Windows Phone 7. L'application consommera un le flux RSS du site MSDN France et affichera le titre des articles dans une liste.

1. Démarrer Visual Studio 2010 Express pour Windows Phone

   > **Remarque:** toutes les étapes décrites dans cet atelier s'appuient sur la version Express de Visual Studio 2010 pour Windows Phone, mais il est tout à fait possible d'utiliser une version complète de Visual Studio 2010 accompagnée des outils de développement pour Windows Phone.

2. Pour ouvrir Microsoft Visual Phone Developer 2010 Express rendez-vous dans le menu Démarrer | Tous les programmes | Microsoft Visual Studio 2010 Express.

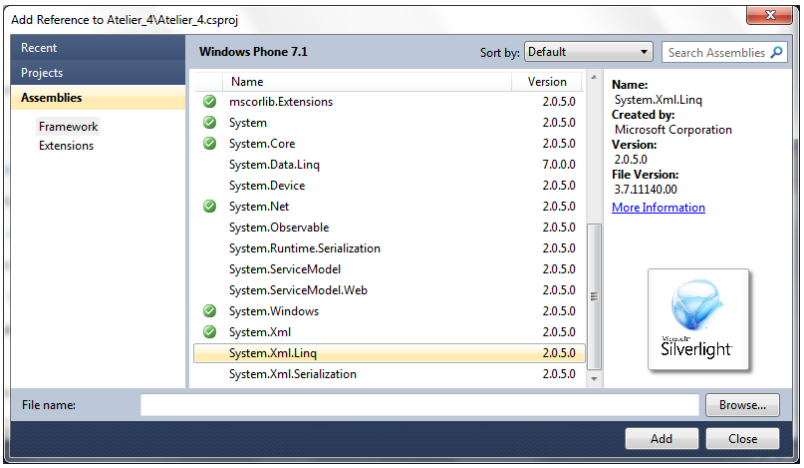
   > **Visual Studio 2010:** Pour ouvrir Visual Studio 2010 rendez-vous dans le menu **Démarrer | Tous les programmes| Microsoft Visual Studio 2010**.

3. Dans le menu File, cliquez sur New Project.

   > **Visual Studio 2010:** Dans le menu File, pointez **New** et cliquez sur **Project**.

4. Dans la fenêtre New Project, choisissez la catégorie Silverlight for Windows Phone et selectionnez Windows Phone Application dans la liste des templates installés.

5. Faîtes un clic droit sur votre projet et choisissez Add reference… puis, dans la fenêtre qui s'ouvre, sélectionnez l'assembly nommée System.Xml.Linq et validez.

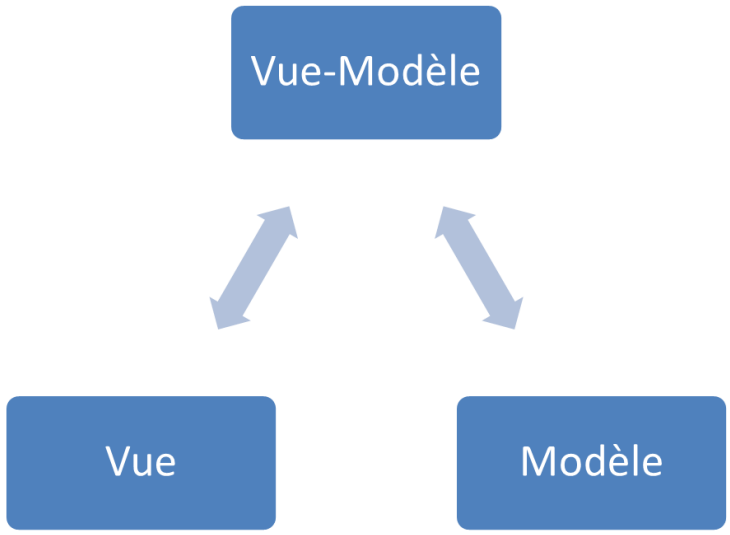


6. A nouveau, faîtes un clic droit sur votre projet et choisissez **Add** puis **Class**. Nommez la **MainPageViewModel** et validez.

   Le pattern M-V-VM (Model-View-ViewModel) permet une séparation entre la couche dédiée aux objets métiers (Model), à l'interface graphique (View), et à la logique (ViewModel).

   Il prend tout son sens avec des technologies telles que Silverlight et WPF où la technologie utilisée pour la création de la couche graphique est le XAML, un langage de balisage abordable pour les non-développeurs qui seraient amenés à travailler sur le projet.

   Le fait d'avoir déporté la complexité de la logique métier dans une couche avec laquelle ils n'ont pas à interagir en utilisant un outil tel que Expression Blend va faciliter leur travail puisqu'ils peuvent se concentrer sur leur mission.



7. Dans la classe nouvellement créée, implémentez l'interface **INotifyPropertyChanged**. Lorsque le ViewModel sera lié à une vue, celle-ci s'abonnera à l'évènement **PropertyChanged**, et lorsque celui-ci sera levé pour une propriété X, la vue mettra automatiquement à jour les contrôles qui sont liés à cette propriété.

   L'interface se trouve dans l'espace de nom **System.ComponentModel**.

```
using System.ComponentModel;


public class MainPageViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(proper
tyName));
    }
```

```
}
```

8. Ajoutez une propriété de type booléen qui sera utilisée pour notifier la vue de l'état de l'application : en train de télécharger des données ou non.

```
private bool _isBusy;

public bool IsBusy
{
    get { return _isBusy; }
    set
    {
        if (_isBusy != value)
        {
            _isBusy = value;
            OnPropertyChanged("IsBusy");
        }
    }
}
```

Pour être en mesure de cacher la barre de chargement lorsque la valeur de **IsBusy** sera à **false**, créez également un converter. Son but sera de transformer une valeur booléenne en valeur de l'énumération **Visibility**.

```
public class BoolToVisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object param
eter, System.Globalization.CultureInfo culture)
    {
        var b = (bool)value;

        return b ? Visibility.Visible : Visibility.Collapsed;
    }

    public object ConvertBack(object value, Type targetType, object p
arameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

9. Sur le ViewModel ; ajoutez une méthode **OnNavigatedTo** selon la signature est définie ci-dessous.

```
public void OnNavigatedTo()
{
}
```

10. Ajoutez une nouvelle classe à votre projet et nommez la **MSDNRssItem**. Elle représentera les objets que vous récupérerez depuis le flux RSS du site MSDN. Ajoutez lui deux propriétés, une pour le titre de l'élement et une pour la date de publication. Une troisième propriété, en lecture seule, représentera la date de publication formatée pour l'affichage.

```
public class MSDNRssItem
```

```
{
    public string Title { get; set; }
    public DateTime Date { get; set; }

    public string FormattedDate
    {
        get
        {
            return Date.ToShortDateString();
        }
    }
}
```

11. Ajoutez une propriété de type **ObservableCollection** à votre ViewModel. Le principe de cette collection est de garder la vue et le ViewModel synchronisé : lorsque le contrôle chargé d'afficher une liste lui est lié, il se mettra automatiquement à jour en cas d'ajout ou de suppression d'éléments.

    Cette classe se trouve dans l'espace de nom **System.Collections.ObjectModel**.

```
using System.Collections.ObjectModel;

namespace Atelier_4
{
    public class MainPageViewModel : INotifyPropertyChanged
    {
        private readonly ObservableCollection<MSDNRssItem> _items =
            new ObservableCollection<MSDNRssItem>();

        public ObservableCollection<MSDNRssItem> Items { get { return
 _items; } }


…
```

12. Complétez la méthode **OnNavigatedTo** afin de simuler un téléchargement de données.

```
public void OnNavigatedTo()
{
    _items.Add(new MSDNRssItem
        {
            Title = "item #1",
            Date = DateTime.Now
        });

    _items.Add(new MSDNRssItem
        {
            Title = "item #2",
            Date = DateTime.Now.AddDays(-3)
        });
}
```

13. Complétez le fichier MainPage.xaml comme ci-dessous pour afficher les données du flux RSS.

```
<phone:PhoneApplicationPage
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:local="clr-namespace:Atelier_4"
    x:Class="Atelier_4.MainPage"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True" d:DataContext="{d:DesignData /S
ampleData/MainPageViewModelSampleData.xaml}">
        <phone:PhoneApplicationPage.Resources>

        <local:BoolToVisibilityConverter x:Key="BoolToVisibilityConv
erter"/>
                <DataTemplate x:Key="MSDNRssItemTemplate">
                        <Grid Width="432"  Margin="0,0,0,24">
                                <Grid.RowDefinitions>
                                        <RowDefinition />
                                        <RowDefinition />
                                </Grid.RowDefinitions>


        <TextBlock Text="{Binding Title}" TextWrapping="Wrap" FontSi
ze="29.333"/>

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Ri
ght" Grid.Row="1">

        <TextBlock Text="publié le " Foreground="{StaticResource Pho
neAccentBrush}" FontFamily="Segoe WP SemiLight" FontSize="18.667" />

        <TextBlock Text="{Binding FormattedDate}" Foreground="{Stati
cResource PhoneAccentBrush}" FontFamily="Segoe WP SemiLight" FontSize
="18.667"/>
                                </StackPanel>
                        </Grid>
                </DataTemplate>
        </phone:PhoneApplicationPage.Resources>

        <phone:PhoneApplicationPage.FontFamily>

        <StaticResource ResourceKey="PhoneFontFamilyNormal"/>
        </phone:PhoneApplicationPage.FontFamily>
        <phone:PhoneApplicationPage.FontSize>
                <StaticResource ResourceKey="PhoneFontSizeNormal"/>
        </phone:PhoneApplicationPage.FontSize>
        <phone:PhoneApplicationPage.Foreground>
                <StaticResource ResourceKey="PhoneForegroundBrush"/>
        </phone:PhoneApplicationPage.Foreground>

    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
```

```
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,
28">
            <TextBlock x:Name="ApplicationTitle" Text="COACHS WINDOWS
 PHONE 7" Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="atelier 4" Margin="9,
-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

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

         <ListBox ItemTemplate="{StaticResource MSDNRssItemTemplate}"
 ItemsSource="{Binding Items}" Margin="12"/>

         <ProgressBar IsIndeterminate="{Binding IsBusy}" Visibility="
{Binding IsBusy, Converter={StaticResource BoolToVisibilityConverter}
}"/>
        </Grid>
    </Grid>

</phone:PhoneApplicationPage>
```

14. Enfin, dans le code behind de la page MainPage.xaml, liéez le ViewModel au **DataContext** et appelez sa méthode **OnNavigatedTo**.

```
public partial class MainPage : PhoneApplicationPage
{
    private readonly MainPageViewModel _vm;

    // Constructor
    public MainPage()
    {
        InitializeComponent();

        _vm = new MainPageViewModel();

        DataContext = _vm;
    }

    protected override void OnNavigatedTo(System.Windows.Navigation.N
avigationEventArgs e)
    {
        base.OnNavigatedTo(e);

        _vm.OnNavigatedTo();
    }
}
```

Executez l'application. Le résultat suivant apparaît alors.

15. Complétez la méthode **OnNavigatedTo** pour aller récupérer les données depuis l'adresse du flux RSS de MSDN et parser ce flux en utilisant le **WebClient** et Linq To Xml. Peuplez ainsi la liste avec les objets générés.

    Ajoutez également les directives using suivantes.

```
using System.Globalization;
using System.Xml.Linq;
using System.Linq;
```

```
public void OnNavigatedTo()
{
    IsBusy = true;

    var webClient = new WebClient();

    webClient.DownloadStringCompleted += (s, e) =>
        {
            var document = XDocument.Parse(e.Result);

            var items = from item in document.Descendants("item").Tak
e(15)
                        select new MSDNRssItem
                        {
                            Title = item.Element("title").Value,
                            Date = DateTime.ParseExact(item.Element("
pubDate").Value, "ddd, d MMM yyyy H:mm:ss GMT", CultureInfo.Invariant
Culture)
                        };

            foreach (var i in items)
```

```
            {
                _items.Add(i);
            }

            IsBusy = false;
        };

    webClient.DownloadStringAsync(new Uri("http://go.microsoft.com/fwlink/?LinkID=110546&clcid=0x40c", UriKind.Absolute));
}
```

Executez l'application, les éléments du flux apparaissent maintenant dans la liste.

16. La particularité du **WebClient** est que le retour de sa méthode **DownloadStringCompleted** est executé dans le même thread que celui qui a servi à l'exécution de la méthode **DownloadStringAsync**. Ainsi, dans l'exemple précédent la création des différents éléments et leur ajout à la liste est effectué sur le thread utilisé par l'interface, ce qui cause un effet de ralentissement de l'application et dégrade l'expérience utilisateur.

    Modifiez la méthode **OnNavigatedTo** comme ce qui suit.

    La méthode **DownloadStringAsync** est maintenant exécutée sur un thread séparé, la callback le sera donc également. Pour pouvoir mettre à jour l'interface, c'est-à-dire ajouter des éléments dans la collection ou modifier la valeur de **IsBusy**, il faut que l'action soit executée sur le thread de l'interface, ce qui est fait au travers de la méthode **BeginInvoke** du **Dispatcher**. L'appel à la méthode **Thread.Sleep** permet de temporiser et de ne pas surcharger le thread de l'interface graphique.

```
public void OnNavigatedTo()
{
    IsBusy = true;

    new Thread(() =>
    {
        var webClient = new WebClient();

        webClient.DownloadStringCompleted += (s, e) =>
            {
                var document = XDocument.Parse(e.Result);

                var items = from item in document.Descendants("item")
.Take(15)
                            select new MSDNRssItem
                            {
                                Title = item.Element("title").Value,
                                Date = DateTime.ParseExact(item.Eleme
nt("pubDate").Value, "ddd, d MMM yyyy H:mm:ss GMT", CultureInfo.Invar
iantCulture)
                            };

                foreach (var i in items)
                {
                    Deployment.Current.Dispatcher.BeginInvoke(((Actio
n<MSDNRssItem>)(item =>
                        {
                            _items.Add(item);
                        })), i);
                    Thread.Sleep(30);
                }

                Deployment.Current.Dispatcher.BeginInvoke(() => IsBus
y = false);
            };

        webClient.DownloadStringAsync(new Uri("http://go.microsoft.co
m/fwlink/?LinkID=110546&clcid=0x40c", UriKind.Absolute));
    }).Start();
}
```

A présent, le téléchargement et l'affichage des données parait totalement fluide.

Exercice 2 – WebRequest et WebResponse

La classe **WebClient** est un wrapper qui utilise en interne les classes **WebRequest** et **WebReponse**. Dans certains cas, il est possible que vous ayez par exemple à personaliser les entêtes HTTP de vos requêtes. Dans cet exercice, vous allez découvrir qu'en utilisant directement ces deux classes, il est possible d'accéder directement à la méchanique interne des requêtes.

1. Remplacez le code existant de la méthode **OnNavigatedTo** par l'extrait de code ci-dessous.

Il faut commencer par créer un objet de type **HttpWebRequest** à partir de l'adresse du flux à récupérer. A ce stade, il est possible de modifier les paramètres de la requête, par exemple en ajoutant des **Credentials** dans la propriété éponyme.
La classe **HttpWebRequest** utilise un modèle de programmation asynchrone. Ainsi, commencez par appelez la méthode **BeginGetResponse**, et dans la callback associée, appelez la méthode **EndGetResponse**. Le flux de la réponse est ensuite accessible via la méthode **GetResponseStream**.

```
public void OnNavigatedTo()
{
    IsBusy = true;

    new Thread(() =>
    {
        var httpWebRequest = HttpWebRequest.CreateHttp("http://go.mic
rosoft.com/fwlink/?LinkID=110546&clcid=0x40c");

        httpWebRequest.BeginGetResponse(cb =>
            {
                var response = httpWebRequest.EndGetResponse(cb);

                using (var stream = response.GetResponseStream())
                {
                    var document = XDocument.Load(stream);

                    var items = from item in document.Descendants("it
em").Take(15)
                                select new MSDNRssItem
                                {
                                    Title = item.Element("title").Val
ue,
                                    Date = DateTime.ParseExact(item.E
lement("pubDate").Value, "ddd, d MMM yyyy H:mm:ss GMT", CultureInfo.I
nvariantCulture)
                                };

                    foreach (var i in items)
                    {
                        Deployment.Current.Dispatcher.BeginInvoke(((A
ction<MSDNRssItem>)(item =>
                        {
                            _items.Add(item);
                        })), i);
                        Thread.Sleep(30);
                    }

                    Deployment.Current.Dispatcher.BeginInvoke(() => I
sBusy = false);
                }
            }, null);
    }).Start();
}
```

Exercice 3 – Les sockets

Dans cet exercice, vous allez apprendre à utiliser les APIs de bas niveau que sont les sockets pour faire communiquer deux applications entres elles.

1. Commencez par créer une nouvelle application Windows Phone 7.
2. Ouvrez le fichier MainPage.xaml et ajoutez lui le contenu ci-dessous.

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

    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

    <TextBox x:Name="textBox" />

    <Button Grid.Row="1" Content="Envoyer !" Click="Button_Click" />

    <TextBlock Grid.Row="2" x:Name="textBlock" />
</Grid>
```

3. Dans le code behind de la page MainPage.xaml, ajoutez la directive using suivante.

```
using System.Net.Sockets;
```

4. L'exemple suivant consiste à rejoindre un canal de diffusion multicast, d'attendre l'arrivée d'un message, et lorsqu'enfin des données sont récupérées dans le buffer, les afficher dans le textblock.
   En paralèle, lorsque l'utilisateur clique sur le bouton, le texte entré dans la zone de saisie est envoyé sur le canal multicast.
   Le fait de rejoindre, d'écouter et d'envoyer des msesages sur un canal multicast est géré par la classe **UdpAnySourceMulticastClient**. Une variante existe pour communiquer avec un interlocuteur précis sur le canal, il s'agit de la classe **UdpSingleSourceMulticastClient**.
   Ajoutez le champ suivant dans le code behind de la page MainPage.xaml.

```
private readonly UdpAnySourceMulticastClient _client;
```

5. Au lancement de l'application, rejoignez le groupe à l'adresse 239.255.255.255 sur le port 26026 via les méthodes asynchrones **BeginJoinGroup** et **EndJoinGroup**. Une fois rejoint, placez vous en attente d'un message via les méthodes asynchrones **BeginReceiveFromGroup** et **EndReceiveFromGroup**. Les données sont reçues sous la forme d'un tableau de byte, pour pouvoir être affichées, transformez en une chaine de caractère en partant du principe qu'elles seront encodées en UTF8.

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

        _client = new UdpAnySourceMulticastClient(IPAddress.Parse("23
9.255.255.255"), 26026);

    new Thread(() =>
    {
        _client.BeginJoinGroup(cbJoinGroup =>
        {
            _client.EndJoinGroup(cbJoinGroup);
```

```
            while (true)
            {
                var receivedBytes = new byte[255];

                try
                {
                    _client.BeginReceiveFromGroup(receivedBytes, 0, r
eceivedBytes.Length, cbReceiveFromGroup =>
                    {
                        IPEndPoint source;
                        _client.EndReceiveFromGroup(cbReceiveFromGrou
p, out source);

                        var received = Encoding.UTF8.GetString(receiv
edBytes, 0, receivedBytes.Length);

                        Deployment.Current.Dispatcher.BeginInvoke(()
=>
                            {
                                textBlock.Text = ("Reçu : " + receive
d.TrimEnd('\0') + " de " + source.ToString());
                            });
                    }, null);
                }
                catch (Exception e)
                {
                        // ...
                }
            }
        }, null);
    }).Start();
}
```

6. Au clic sur le bouton, encodez en UTF8 le contenu de la textbox et transmettez-le via la méthode **BeginSendToGroup**.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var data = Encoding.UTF8.GetBytes(textBox.Text);
    _client.BeginSendToGroup(data, 0, data.Length, cbSendToGroup =>{}
, null);
}
```

Exercice 4 – Téléchargement en tâche de fond

Il est possible d'effectuer un téléchargement ou un envoi de données qui seront exécutés en tâche de fond alors même que l'utilisateur n'est plus en train de se servir de votre application. Dans cet exercice, vous allez apprendre à créer une nouvelle tâche auprès du Background Transfer Service du téléphone, et à requêter ce même service pour afficher l'état d'une tâche.

1. Commencez par créer un nouveau projet de type **Windows Phone Application** dans la catégorie **Silverlight for Windows Phone**.
2. Modifiez la page MainPage.xaml comme ci-dessous.

```
<Grid x:Name="LayoutRoot" Background="Transparent">
```

```
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <!--
TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="COACHS WINDOWS PHO
NE 7" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Atelier 4" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <ItemsControl ItemsSource="{Binding Requests}">
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <Grid Margin="0,0,0,6" Width="450">
                        <Grid.RowDefinitions>
                            <RowDefinition />
                            <RowDefinition />
                            <RowDefinition Height="Auto" />
                        </Grid.RowDefinitions>
                        <TextBlock Text="{Binding Tag}" />
                        <TextBlock Grid.Row="1" Text="{Binding Status
}" />
                        <ProgressBar Grid.Row="2" Visibility="{Bindin
g ProgressBarVisibility}" Value="{Binding Progression}" />
                    </Grid>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
        </ItemsControl>

        <Border Grid.Row="1" BorderThickness="4" BorderBrush="{Static
Resource PhoneForegroundBrush}">
            <Grid Margin="6">
                <Grid.RowDefinitions>
                    <RowDefinition />
                    <RowDefinition />
                    <RowDefinition />
                </Grid.RowDefinitions>
                <TextBox x:Name="textBox" />
                <Button x:Name="button" Grid.Row="2" Content="démarre
r le transfert" Click="button_Click" />
            </Grid>
        </Border>
    </Grid>
</Grid>
```

3. L'utilisation du service se fait au travers de la classe statique **BackgroundTransferService** de l'espace de nom

Microsoft.Phone.BackgroundTransfert. Avec ce service, les requêtes sont représentées par la classe **BackgroundTransfertRequest**.
Au clic sur le bouton, vous enverrez une nouvelle requête de téléchargement au service.
Les différentes requêtes seront encapsulées dans une classe **RequestViewModel** afin d'afficher plus facilement des informations les concernant, à savoir leur status et le pourcentage de progression de la requête. Ces deux propriétés sont calculables car les objets de type **BackgroundTransfertRequest** sont en mesure de lever deux évènements, **TransferStatusChanged** et **TransferProgressChanged**.

```
public class RequestViewModel : INotifyPropertyChanged
{
    private BackgroundTransferRequest _innerRequest;

    public RequestViewModel(BackgroundTransferRequest innerRequest)
    {
        _innerRequest = innerRequest;

        UpdateStatus();
        UpdateProgression();

        _innerRequest.TransferStatusChanged += new EventHandler<Backg
roundTransferEventArgs>(_innerRequest_TransferStatusChanged);

        _innerRequest.TransferProgressChanged += new EventHandler<Bac
kgroundTransferEventArgs>(_innerRequest_TransferProgressChanged);
    }

    void _innerRequest_TransferStatusChanged(object sender, Backgroun
dTransferEventArgs e)
    {
        UpdateStatus();
    }

    private void UpdateStatus()
    {
        if (_innerRequest.TransferStatus == TransferStatus.Completed)
        {
            ProgressBarVisibility = Visibility.Collapsed;
            Status = "Terminé";
        }
        else if (_innerRequest.TransferStatus == TransferStatus.Trans
ferring)
        {
            ProgressBarVisibility = Visibility.Visible;
            Status = "Transfert en cours";
        }
        else
        {
            ProgressBarVisibility = Visibility.Collapsed;
            Status = "En attente";
        }
    }

    void _innerRequest_TransferProgressChanged(object sender, Backgro
undTransferEventArgs e)
    {
```

```
        UpdateProgression();
    }

    private void UpdateProgression()
    {
        if(_innerRequest.TotalBytesToReceive == 0)
            Progression = 0;
        else
            Progression = (int)((_innerRequest.BytesReceived * 100) /
_innerRequest.TotalBytesToReceive);
    }

    private int _progression;

    public int Progression
    {
        get { return _progression; }
        set
        {
            _progression = value;
            OnPropertyChanged("Progression");
        }
    }

    private string _status;

    public string Status
    {
        get { return _status; }
        set
        {
            _status = value;
            OnPropertyChanged("Status");
        }
    }

    private Visibility _progressBarVisibility;

    public Visibility ProgressBarVisibility
    {
        get { return _progressBarVisibility; }
        set
        {
            _progressBarVisibility = value;
            OnPropertyChanged("ProgressBarVisibility");
        }
    }

    public string Tag
    {
        get
        {
            return _innerRequest.Tag;
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
```

```
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(proper
tyName));
    }
}
```

4. Dans la classe **MainPage**, ajoutez une **ObservableCollection** d'objets de type **RequestViewModel**.

```
private readonly ObservableCollection<RequestViewModel> _requests
        = new ObservableCollection<RequestViewModel>();

public ObservableCollection<RequestViewModel> Requests { get { return
 _requests; } }
```

5. Au lancement de l'application, récupérez les requêtes existantes qui seraient en attente ou en cours de traitement par le service de téléchargement via sa propriété **Requests**. Ajoutez les à la collection de **RequestViewModel**.

```
foreach (var r in BackgroundTransferService.Requests)
{
    Requests.Add(new RequestViewModel(r));
}
```

6. Au clic sur le bouton, créez une nouvelle requête. Passez la ensuite à la méthode Add du service pour qu'elle soit traitée par celui-ci. Afin de ne pas détériorer l'expérience utilisateur, passez-la au service dans un thread séparé.
   Les fichiers à télécharger doivent impérativement l'être dans le dossier /shared/transfers de l'isolated storage.

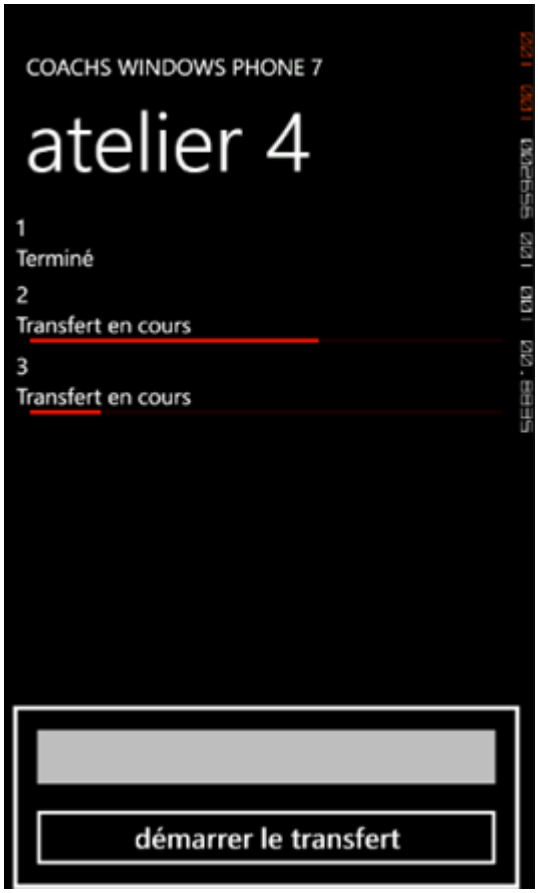
```
private void button_Click(object sender, RoutedEventArgs e)
{
    var request = new BackgroundTransferRequest(new Uri("http://downl
oad.thinkbroadband.com/50MB.zip", UriKind.Absolute));

    request.DownloadLocation = new Uri("shared/transfers/" + textBox.
Text, UriKind.RelativeOrAbsolute);
    request.Tag = textBox.Text;

        _requests.Add(new RequestViewModel(request));

    new Thread(() =>
    {
        BackgroundTransferService.Add(request);
    }).Start();
}
```

7. Vous pouvez vous servir du **BackgroundTransfertService** pour envoyer un fichier simplement en changent le verbe HTTP associé à la requête via la propriété **Method**.
8. Il y a quelques règles à respecter concernant l'utilisation du **BackgroundTransferService** :
   a. Dans le cas de d'envoie d'un fichier, la taille de celui ne doit pas excéder 5mb.

b. La taille maximale d'un fichier qui peut être téléchargé en utilisant une connexion de données est de 20mb.
c. La taille maximale d'un fichier qui peut être téléchargé en Wifi sans que le téléphone soit branché à une source d'alimentation est de 100mb.
d. Il ne peut y avoir que deux transferts en simultanés, si d'autres sont soumis au service, ils sont automatiquement placés dans une file d'attente.
e. Le service de téléchargement supporte au maximum 5 fichiers en téléchargement et en file d'attente par application.

Il est possible d'établir des préférences sur le type de connection à utiliser pour télécharger un fichier via la propriété TransferPreferences.

9. Executez l'application et lancez le téléchargement de quelques fichiers.



10. Quittez l'application mais ne fermez pas l'émulateur. Dans le fichier WMAppManifest.xml, modifiez le GUID de l'application puis relancez-la. Pour le téléphone, il s'agit maintenant d'une nouvelle application, le service ne liste donc pas les téléchargements que vous avez précédemment lancés.
11. A nouveau, quittez l'application mais ne fermez pas l'émulateur. Dans le fichier WMAppManifest.xml, restaurez l'ancien GUID de l'application et démarrez-la. Les téléchargements sont à nouveau listés.

Exercice 5 – Découverte de l'isolated storage

Dans cet exercice, vous allez découvrir comment utiliser le stockage du téléphone pour persister des données et être ensuite capable de les récupérer dans votre application.

1. Commencez par créer un nouveau projet de type **Windows Phone Application**.
2. Modifiez la page MainPage.xaml comme ci-dessous.

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="COACHS WINDOWS PHONE 7"
Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Atelier 4" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

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

        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"  />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Border Margin="12" BorderThickness="3" BorderBrush="{StaticResourc
e PhoneForegroundBrush}">
            <StackPanel Margin="6">
                <TextBlock Margin="12,0,0,0" Text="Prénom" />
                <TextBox />
                <TextBlock Margin="12,0,0,0" Text="Nom" />
                <TextBox />
            </StackPanel>
        </Border>

        <Button Grid.Row="1" Content="Sauvegarder" />

        <Button Grid.Row="2" Content="Charger" />

    </Grid>
</Grid>
```

3. Créez un ViewModel pour cette page. Celui la contiendra deux propriétés qui seront bindées aux textbox de la page. Ajoutez également des méthodes Save et Load qui seront plus tard liées aux boutons correspondants.

```
public class MainPageViewModel : INotifyPropertyChanged
{
    private string _nom;

    public string Nom
    {
        get { return _nom; }
        set
```

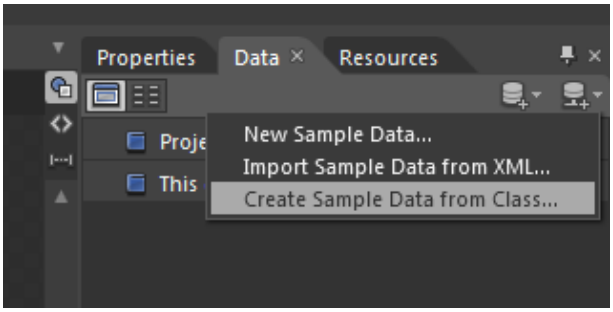
```
        {
            _nom = value;
            OnPropertyChanged("Nom");
        }
    }

    private string _prenom;

    public string Prenom
    {
        get { return _prenom; }
        set
        {
            _prenom = value;
            OnPropertyChanged("Prenom");
        }
    }

    public MainPageViewModel()
    {
    }

    public void Save()
    {
    }

    public void Load()
    {
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName
));
        }
    }
}
```

4. Définissez le **DataContext** de la page avec le view model que vous venez de créer.

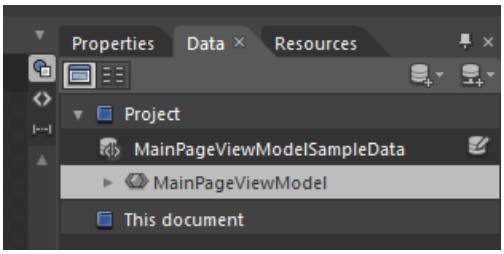
```
public partial class MainPage : PhoneApplicationPage
{
    // Constructor
    public MainPage()
    {
        InitializeComponent();

        DataContext = new MainPageViewModel();
    }
}
```

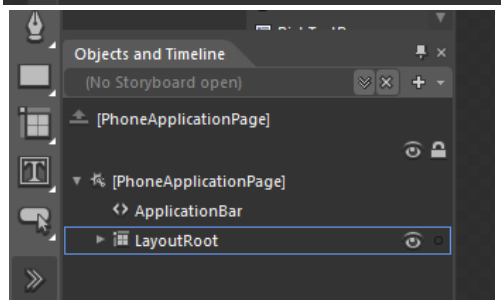
5. Compilez la solution, faîtes un clic droit sur le projet dans l'explorateur de solution et choisissez de l'ouvrir avec Expression Blend.

6. Une fois Blend ouvert, dans l'onglet Data, cliquez sur « Create sample data », et choisissez l'option « Create sample data from class… ».



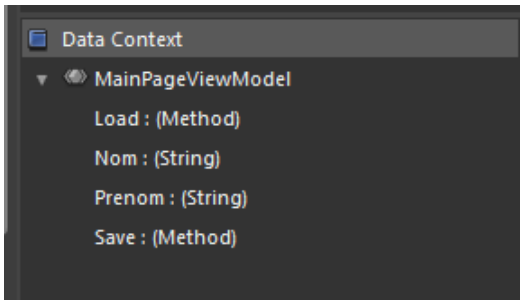
7. Dans la fenêtre qui s'ouvre, cherchez la classe **MainPageViewModel** et validez en cliquant sur OK. Expression Blend va alors générer un fichier contenant des données exemples qui permettront de designer plus facilement les vues de l'application.
8. Dans l'onglet Data, faîtes glissez l'objet nommé **MainPageViewModel** jusqu'à l'objet **PhoneApplicationPage** de l'onglet « Objects and Timeline ».



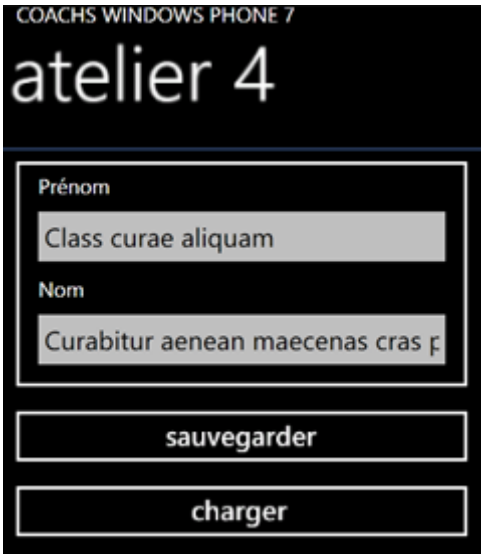


Blend a alors défini le DataContext « design-time », c'est-à-dire le DataContext à utiliser lorsque vous éditez votre application dans Expression Blend (ou n'importe quel auter designer), comme étant une instance des données exemples que vous avez précédemment générés.

9. Dans l'onglet **DataContext**, vous retrouverez maintenant les différentes propriétés et méthodes du view model.



10. Faites glissez la propriété Prenom sur la première textbox et la propriété Nom sur la seconde textbox. Blend va alors automatiquement générer les bindings pour ces deux textbox.



11. Faîtes à présent glisser la méthode **Save** vers le bouton « Sauvegarder » et la méthode **Load** vers le bouton « Charger ». Là encore, Blend va automatiquement utiliser pour ces deux boutons une behavior appelée **CallMethodAction**.
    Les behaviors sont des composants réutilisables qui permettent d'étendre le fonctionnement des controles afin de faciliter le travail des designers.
    La behavior **CallMethodAction** permet d'appeler une méthode d'un objet (dans notre cas, du **DataContext**) à condition que celle ci ne prenne pas de paramètres.
    Le code de la page MainPage.xaml devrait à present ressembler à celui de l'extrait de code ci dessous.

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="COACHS WINDOWS PHONE 7"
Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Atelier 4" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

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

        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"  />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Border Margin="12" BorderThickness="3" BorderBrush="{StaticResourc
e PhoneForegroundBrush}">
            <StackPanel Margin="6">
                <TextBlock Margin="12,0,0,0" Text="Prénom" />
                <TextBox Text="{Binding Prenom, Mode=TwoWay}" />
                <TextBlock Margin="12,0,0,0" Text="Nom" />
                <TextBox Text="{Binding Nom, Mode=TwoWay}" />
            </StackPanel>
```

```
        </Border>

        <Button Grid.Row="1" Content="sauvegarder" >
        <i:Interaction.Triggers>
                <i:EventTrigger EventName="Click">

        <ec:CallMethodAction MethodName="Save" TargetObject="{Binding}"/>
                </i:EventTrigger>
        </i:Interaction.Triggers>
        </Button>

        <Button Grid.Row="2" Content="charger" >
        <i:Interaction.Triggers>
                <i:EventTrigger EventName="Click">

        <ec:CallMethodAction MethodName="Load" TargetObject="{Binding}"/>
                </i:EventTrigger>
        </i:Interaction.Triggers>
        </Button>

    </Grid>
</Grid>
```

12. La première chose à faire pour pouvoir utiliser l'isolated storage est de récupérer une reference vers le conteneur dédié à votre application. Cela se fait via la méthode statique **GetUserStoreForApplication** de la classe **IsolatedStorageFile**.
    A partir d'une instance de la classe **IsolatedStorageFile**, il est possible de manipuler repertoires et fichiers. L'extrait de code suivant récupère le storage de l'application et il ouvre un flux vers un fichier nommé "data.dat". La valeur **Create** de l'énumération **FileMode** est telle que si le fichier n'existe pas, il est crée, sinon il est tronqué.

```
public void Save()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        using (var isolatedStorageFileStream = isolatedStorage.OpenFile("da
ta.dat", System.IO.FileMode.Create))
        {
        }
    }
}
```

13. Pour sérialiser du contenu dans un flux de manière optimale, vous pouvez utiliser la classe **BInaryWriter**. Celle ci possède une méthode **Write** avec des surcharges pour la quasi totalité des structures de base du framework.
    Le code suivant écrira dans le flux les deux propriétés Prenom et Nom.

```
public void Save()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        using (var isolatedStorageFileStream = isolatedStorage.OpenFile("da
ta.dat", System.IO.FileMode.Create))
        {
```

```
            using (var binaryWriter = new BinaryWriter(isolatedStorageFileS
tream))
            {
                binaryWriter.Write(Prenom);
                binaryWriter.Write(Nom);
            }
        }
    }
}
```

14. Pour lire le contenu sérialisé, utilisez la classe **BinaryReader**. Celle ci possède des méthodes pour tous les types de base : entiers, chaines de caractères, etc. Attention ! L'odre dans lequel les propriétés sont désérialisées doit être le même que l'ordre dans lequel elles ont été sérialisées.

```
public void Load()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        if (isolatedStorage.FileExists("data.dat"))
        {
            using (var isolatedStorageFileStream = isolatedStorage.OpenFile
("data.dat", System.IO.FileMode.Open))
            {
                using (var binaryReader = new BinaryReader(isolatedStorageF
ileStream))
                {
                    Prenom = binaryReader.ReadString();
                    Nom = binaryReader.ReadString();
                }
            }
        }
    }
}
```

15. Executez l'application. Complétez les deux champs et cliquez sur le bouton "Sauvegarder".
16. Quittez l'application.
17. Relancez l'application et cliquez sur le bouton "Charger". Les deux champs retrouvent les valeurs que vous aviez rentrées précédemment.

18. Généralement, lors du développement d'une application, vous préfèrerez utiliser des classes métiers pour encapsuler les propriétés telles que **Prenom** et **Nom**.
    Ajoutez donc une classe **Customer** à votre projet avec ces deux propriétés.

```
public class Customer : INotifyPropertyChanged
{
    private string _nom;

    public string Nom
    {
        get { return _nom; }
        set
        {
            _nom = value;
            OnPropertyChanged("Nom");
        }
    }

    private string _prenom;

    public string Prenom
    {
        get { return _prenom; }
        set
        {
            _prenom = value;
            OnPropertyChanged("Prenom");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
```

```
    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName
));
        }
    }
}
```

19. Modifiez la classe **MainPageViewModel** afin d'utiliser un objet de type Customer plutôt que les propriétés **Prenom** et **Nom**.

```
public class MainPageViewModel : INotifyPropertyChanged
{
    private Customer _customer;

    public Customer Customer
    {
        get { return _customer; }
        set
        {
            _customer = value;
            OnPropertyChanged("Customer");
        }
    }

    public MainPageViewModel()
    {
        Customer = new Customer();
    }

    …
}
```

20. Modifiez le **DataContext** du stackpanel qui contient les deux textbox afin qu'il utilise l'objet de type **Customer**.

```
<Border Margin="12" BorderThickness="3" BorderBrush="{StaticResource PhoneF
oregroundBrush}">
    <StackPanel Margin="6" DataContext="{Binding Customer}">
        <TextBlock Margin="12,0,0,0" Text="Prénom" />
        <TextBox Text="{Binding Prenom, Mode=TwoWay}" />
        <TextBlock Margin="12,0,0,0" Text="Nom" />
        <TextBox Text="{Binding Nom, Mode=TwoWay}" />
    </StackPanel>
</Border>
```

21. Vous pouvez déporter le détail de la sérialisation d'un objet de type **Customer** en créant des méthodes d'extensions pour **BinaryReader** et **BinaryWriter**.

```
public static class BinaryWriterExtension
{
    public static void Write(this BinaryWriter binaryWriter, Customer custo
mer)
    {
```

```
        binaryWriter.Write(customer.Prenom);
        binaryWriter.Write(customer.Nom);
    }
}

public static class BinaryReaderExtension
{
    public static Customer ReadCustomer(this BinaryReader binaryReader)
    {
        var customer = new Customer();
        customer.Prenom = binaryReader.ReadString();
        customer.Nom = binaryReader.ReadString();
        return customer;
    }
}
```

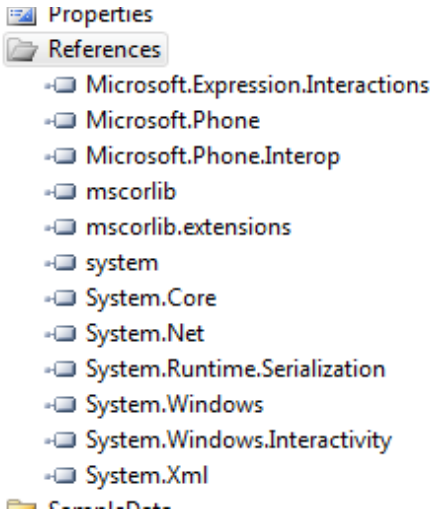
Et vous pouvez les utiliser dans le ViewModel :

```
public void Save()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        using (var isolatedStorageFileStream = isolatedStorage.OpenFile("da
ta.dat", System.IO.FileMode.Create))
        {
            using (var binaryWriter = new BinaryWriter(isolatedStorageFileS
tream))
            {
                binaryWriter.Write(Customer);
            }
        }
    }
}

public void Load()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        if (isolatedStorage.FileExists("data.dat"))
        {
            using (var isolatedStorageFileStream = isolatedStorage.OpenFile
("data.dat", System.IO.FileMode.Open))
            {
                using (var binaryReader = new BinaryReader(isolatedStorageF
ileStream))
                {
                    Customer = binaryReader.ReadCustomer();
                }
            }
        }
    }
}
```

22. Pour sérialiser plus facilement des objets, il est possible d'utiliser le **DataContractSerializer**. Celui ci parcourera automatiquement les différentes propriétés sérialisables de vos objets et s'occupera de tout.

Pour être utilisable avec le **DataContractSerializer**, il faut décorer ses entités avec l'attribut **DataContract**. C'est le cas notamment des classes générées par l'utilitaire d'ajout de reference à un service web.

Commencez par ajouter une référence à **System.Runtime.Serialization**.



Ajoutez ensuite la directive using suivante.

```
using System.Runtime.Serialization;
```

Modifiez votre classe **Customer** comme ci dessous. Décorez les propriétés à sérialiser avec l'attribut **DataMember**. Pour vous assurer qu'une propriété ne sera pas sérialisée, décorez-la de l'attribut **IgnoreDataMember**.

```
[DataContract]
public class Customer : INotifyPropertyChanged
{
    private string _nom;

    [DataMember]
    public string Nom
    {
        get { return _nom; }
        set
        {
            _nom = value;
            OnPropertyChanged("Nom");
        }
    }

    private string _prenom;

    [IgnoreDataMember]
    public string Prenom
    {
        get { return _prenom; }
        set
        {
            _prenom = value;
            OnPropertyChanged("Prenom");
```

```
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName
));
        }
    }
}
```

23. Utilisez la classe **DataContractSerialiser** et ses méthodes **WriteObject** et **ReadObject**, respectivement pour sérialiser et pour désérialiser un objet. Dans le cas de la deserialization, le résultat est boxé, il faut donc le caster pour obtenir un objet de type **Customer**.

```
public void Save()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        using (var isolatedStorageFileStream = isolatedStorage.OpenFile("da
ta.dat", System.IO.FileMode.Create))
        {
            var dataContractSerilizer = new DataContractSerializer(typeof(C
ustomer));
            dataContractSerilizer.WriteObject(isolatedStorageFileStream, Cu
stomer);
        }
    }
}

public void Load()
{
    using (var isolatedStorage = IsolatedStorageFile.GetUserStoreForApplica
tion())
    {
        if (isolatedStorage.FileExists("data.dat"))
        {
            using (var isolatedStorageFileStream = isolatedStorage.OpenFile
("data.dat", System.IO.FileMode.Open))
            {
                var dataContractSerilizer = new DataContractSerializer(type
of(Customer));
                Customer = dataContractSerilizer.ReadObject(isolatedStorage
FileStream) as Customer;
            }
        }
    }
}
```

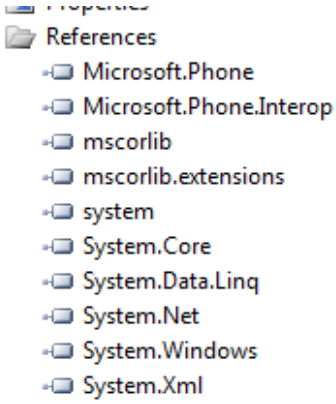
24. Executez l'application. Saisissez des informations et quittez la. En la relançant à nouveau, et en chargeant les données, seule une des deux propriétés est restaurée. Ce comportement

est bien celui attendu puisque l'autre propriété avec été décorée avec l'attribut **IgnoreDataMember**.

Exercice 6 – Utilisation d'une base de données locale

Dans cet exercice, vous allez apprendre à utiliser SQL CE pour windows phone, à générer une base, la provisionner en données puis à requêter ces données.

1. Créez un nouveau projet de type **Windows Phone Application**.
2. Ajoutez une référence à **System.Data.Linq**.



3. Ajoutez une classe **Customer** à votre projet. Cette classe sera mappée à une table éponyme dans la base de données. Pour que cela soit possible, décorez-la avec l'attribut **Table** situé dans l'espace de nom **System.Data.Linq.Mappings**. De même, pour que ses propriétés soient mappées aux colonnes de la table, vous devez les décorer de l'attribut **Column**. Dans le cas de l'id, il est également nécessaire de préciser qu'il s'sagit de la clé primaire de la table. Notez qu'il est également possible de préciser un nom différent pour la colonne via la propriété **Name**, qu'il est possible de spécifier qu'il s'agit d'un champ nullable via la propriété

CanBeNull, ou encore qu'il est possible de spécifier explicitement le type SQL que la colonne doit utiliser grâce à la propriété **DbType**. Ce dernier point est relativement important, puisque par défaut, les chaines de caractères sont mappées à un type **varchar** avec une taille maximale fixée à 4000 bytes. Pour de plus grandes chaines de caractères, spécifiez que vous voulez que la propriété soit mappée à une colonne de type **ntext**.

```
[Table]
public class Customer
{
    [Column(IsPrimaryKey=true)]
    public int Id { get; set; }

    [Column]
    public string Name { get; set; }

    [Column]
    public string Country { get; set; }
}
```

4. Modifiez la page MainPage.xaml comme ci-dessous.

```
        <phone:PhoneApplicationPage.Resources>
                <DataTemplate x:Key="CustomerTemplate">

        <StackPanel Margin="12,0,0,6" Orientation="Horizontal">

        <TextBlock TextWrapping="Wrap" Text="{Binding Id}" FontSize="26.667
"/>
                                    <TextBlock TextWrapping="Wrap" Text=" -
 " FontSize="26.667"/>

        <TextBlock Text="{Binding Name}" FontSize="26.667"/>
                        </StackPanel>
                </DataTemplate>
        </phone:PhoneApplicationPage.Resources>

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-
->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="COACHS WINDOWS PHONE
 7" Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="atelier 4" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

        <ItemsControl ItemTemplate="{StaticResource CustomerTemplate}" Item
sSource="{Binding Customers}"/></Grid>
    </Grid>
```

5. Ajoutez une classe **MainPageViewModel** et utilisez comme **DataContext** de la page. Dans le ViewModel, ajoutez une méthode **OnNavigatedTo** que vous appelerez lorsque l'évènement du même nom sera levé pour la page **MainPage**.

```
public partial class MainPage : PhoneApplicationPage
{
    private readonly MainPageViewModel _vm;

    // Constructor
    public MainPage()
    {
        InitializeComponent();

        _vm = new MainPageViewModel();

        DataContext = _vm;
    }

    protected override void OnNavigatedTo(System.Windows.Navigation.Navigat
ionEventArgs e)
    {
        base.OnNavigatedTo(e);

        _vm.OnNavigatedTo();
    }
}
```

6. Pour représenter votre base de données, vous devez créer une classe dérivant de la classe **DataContext** de l'espace de nom **System.Data.Linq**. Cette classe contiendra un champ public pour chaque table que doit contenir votre base. Ce champ est de type **Table<T>** où **T** représente une entité marquée par l'attribut **Table**. Cette classe a également besoin de recevoir une chaine de connexion qu'elle devra passer au constructeur de la classe **DataContext**.

```
public class MyDataContext : DataContext
{
    public MyDataContext(string connectionString)
        : base(connectionString)
    {
    }

    public Table<Customer> Customers;
}
```

7. Dans la classe **MainPageViewModel**, ajoutez un champ de type **MyDataContext**.

```
private readonly MyDataContext _dataContext;
```

8. Dans le constructeur de la classe **MainPageViewModel**, initialisez l'objet de type **MyDataContext**. La chaine de connexion doit correspondre au chemin vers le fichier .sdf

(SQL CE) de la base de données. Le fait qu'elle soit préfixée de « isostore » signifie que la base est stockée dans l'isolated storage. Si la base de données est livrée en tant que contenu d'application dans le .xap, elle est accessible en préfixant la chaine de connexion de « appdata » :

```
   public MainPageViewModel()
        {
            _dataContext = new MyDataContext("Data Source='isostore:/custom
ers.sdf';");
        }
```

9. Lorsque la base de données est stockée dans l'isolated storage, il est possible de tester son existence via la méthode **DatabaseExists**. Dans le cas où elle n'existe pas, sa création se fera au travers de la méthode **CreateDatabase**. Enfin, l'ajout d'élements dans une table est possible en utilisant la méthode **InsertOnSubmit** des objets **Table<T>**. Dans notre cas, il s'agit donc de la méthode **InsertOnSubmit** de la propriété **Customers**. La validation pour que les données soit effectivement sauvegardées dans la base est possible grâce à la méthode **SubmitChanges**.

```
public MainPageViewModel()
{
    _dataContext = new MyDataContext("Data Source='isostore:/customers.sdf'
;");

    if(!_dataContext.DatabaseExists())
    {
        _dataContext.CreateDatabase();

        _dataContext.Customers.InsertOnSubmit(new Customer { Id = 1, Name =
 "Pierre", Country = "France" });
        _dataContext.Customers.InsertOnSubmit(new Customer { Id = 2, Name =
 "Bob", Country = "USA" });
        _dataContext.Customers.InsertOnSubmit(new Customer { Id = 3, Name =
 "Luc", Country = "France" });

        _dataContext.SubmitChanges();
    }
}
```

10. Ajoutez une **ObservableCollection** de **Customer** à la classe **MainPageViewModel**.

```
private readonly ObservableCollection<Customer> _customers
    = new ObservableCollection<Customer>();

public ObservableCollection<Customer> Customers { get { return _customers;
} }
```

11. Dans la méthode **OnNavigatedTo** de la classe **MainPageViewModel**, chargez dans un thread séparé les clients résidant en France. Insérez ensuite ces clients dans la collection que vous venez de créer.

```
public void OnNavigatedTo()
{
    new Thread(() =>
    {
```

```
        var customers = from c in _dataContext.Customers
                        where c.Country == "France"
                        select c;

        foreach (var c in customers)
        {
            Deployment.Current.Dispatcher.BeginInvoke((Action<Customer>)(cu
stomer =>
                {
                    _customers.Add(customer);
                }), c);
        }
    }).Start();
}
```

12. Executez l'application, les clients que vous aviez associés à la France apparaissent dans la liste.

13. Créez une nouvelle classe **Country** avec la structure suivante.

```
[Table]
public class Country
{
    [Column(IsPrimaryKey = true)]
    public int Id { get; set; }

    [Column]
    public string Name { get; set; }
}
```

14. Ajoutez une table de **Country** dans la classe **MyDataContext**.

```
public class MyDataContext : DataContext
{
    public MyDataContext(string connectionString)
```

```
        : base(connectionString)
    {
    }

    public Table<Customer> Customers;

    public Table<Country> Countries;
}
```

15. Dans la classe **Customer**, supprimez la propriété de type de chaine de caractère et remplacez-la par une propriété de type **Country**. Cette propriété n'aura pas besoin d'attribut **Column**.
    Pour modéliser correctement la relation, il est nécessaire d'ajouter une propriété **CountryId** de type int qui contiendra l'id du pays lié, c'est cette propriété qui sera décorée de l'attribut **Column**.
    Ajoutez un champ de type **EntityRef<T>** où **T** est de type **Country**. C'est lui qui va permettre de tracker l'entité de la relation plus facilement.
    Sur la propriété **Country**, ajoutez un attribut **Association** qui spécifie le champ à utiliser pour le tracking, la propriété correspondant à la clé étrange et celle correspondant à la clé primaire dans la table **Country**.

```
[Table]
public class Customer
{
    [Column(IsPrimaryKey=true)]
    public int Id { get; set; }

    [Column]
    public string Name { get; set; }

    [Column]
    public int CountryId { get; set; }

    private EntityRef<Country> _countryRef = new EntityRef<Country>();

    [Association(Name="FK_Customer_Country", Storage="_countryRef", OtherKe
y = "Id", ThisKey = "CountryId", IsForeignKey = true)]
    public Country Country
    {
        get { return _countryRef.Entity; }
        set
        {
            _countryRef.Entity = value;
        }
    }
}
```

16. Modifiez le constructeur de la classe **MainPageViewModel** afin de créer des objets **Country** et de lier les entités **Customer** à ces objets.

```
public MainPageViewModel()
{
    _dataContext = new MyDataContext("Data Source='isostore:/customers.sdf'
;");
```

```
    if(!_dataContext.DatabaseExists())
    {
        _dataContext.CreateDatabase();

        var france = new Country { Id = 1, Name = "France" };
        var usa = new Country { Id = 2, Name = "USA" };

        _dataContext.Countries.InsertOnSubmit(france);
        _dataContext.Countries.InsertOnSubmit(usa);

        _dataContext.Customers.InsertOnSubmit(new Customer { Id = 1, Name =
 "Pierre", Country = france });
        _dataContext.Customers.InsertOnSubmit(new Customer { Id = 2, Name =
 "Bob", Country = usa });
        _dataContext.Customers.InsertOnSubmit(new Customer { Id = 3, Name =
 "Luc", Country = france });

        _dataContext.SubmitChanges();
    }
}
```

17. Modifiez également la requête dans la méthode **OnNavigatedTo** afin de prendre en compte la propriété de l'association.

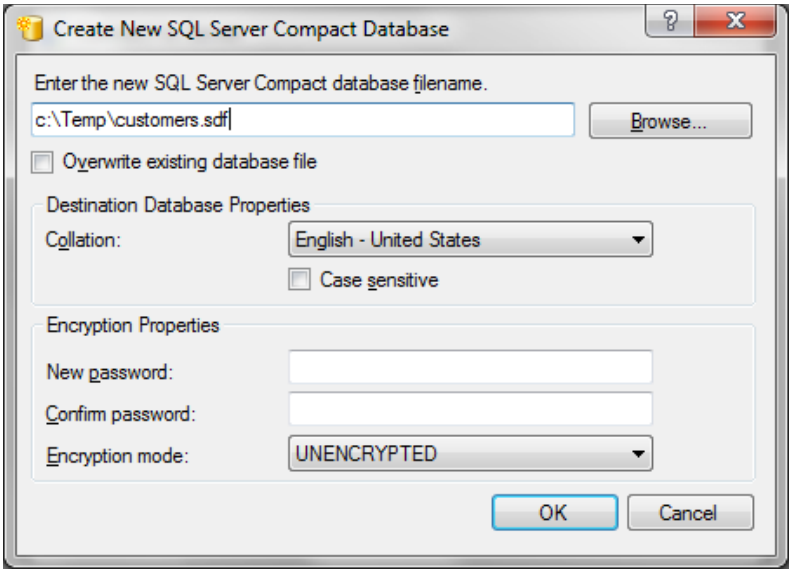
```
public void OnNavigatedTo()
{
    new Thread(() =>
    {
        var customers = (from c in _dataContext.Customers
                         where c.Country.Name == "France"
                         select c).ToList();

        foreach (var c in customers)
        {
            Deployment.Current.Dispatcher.BeginInvoke((Action<Customer>)(cu
stomer =>
                {
                    _customers.Add(customer);
                }), c);
        }
    }).Start();
}
```

Exercice 7 – Utilisation d'une base de données SQL CE existante dans une application Windows Phone 7

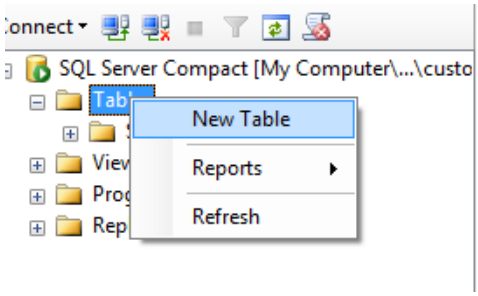
Dans certains scénarios, il est nécessaire d'utiliser une base de données existante qui sera distribuée avec une application Windows Phone 7.

1. Ouvrez SQL Server Management Studio. Dans la fenêtre de connexion à un serveur, choisissez en type de serveur « SQL Server Compact ».

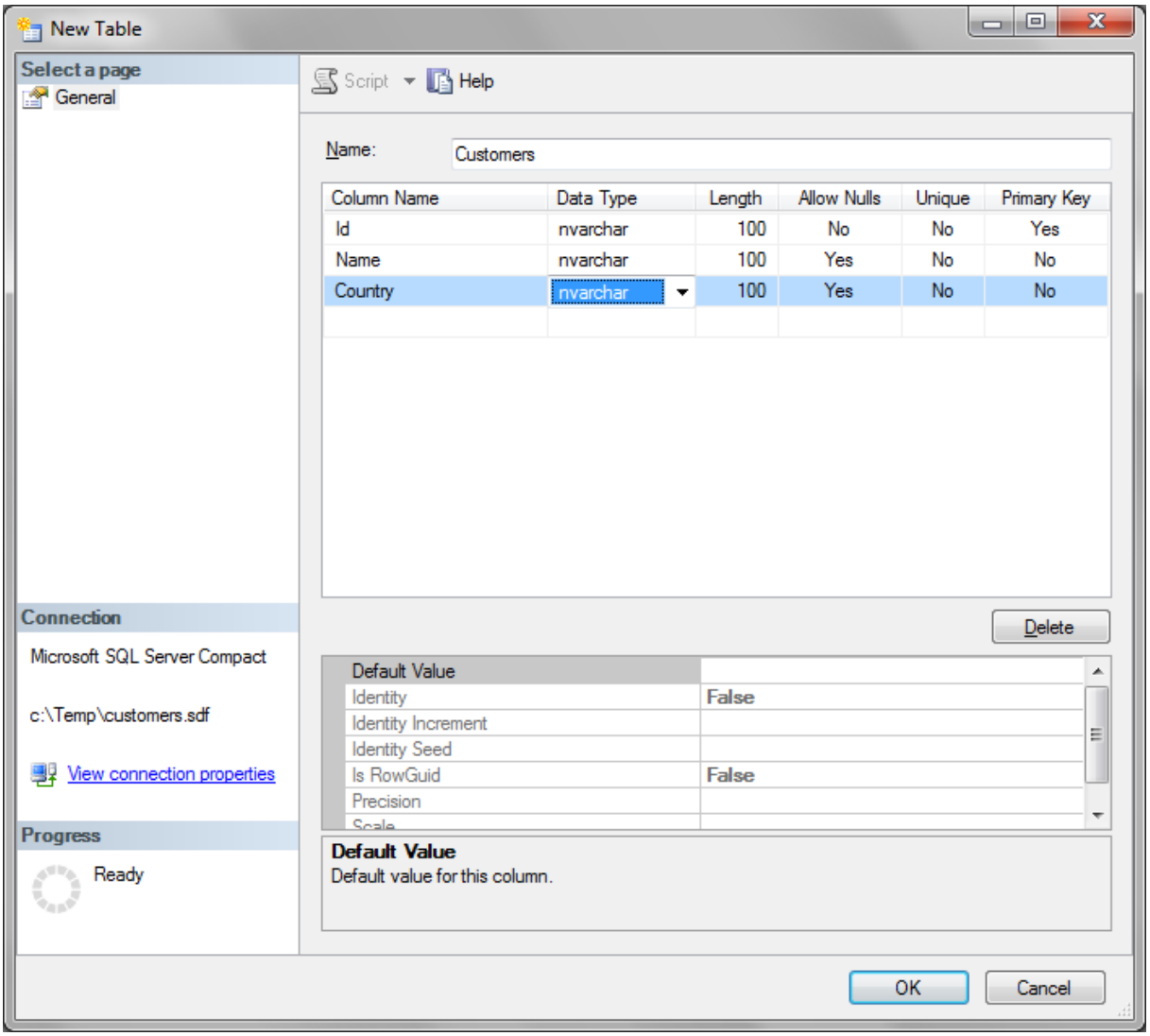
2. Dans la liste déroulante du fichier de base de données, choisissez de créer une nouvelle base.



3. Une fois connecté à la base de données, créez une nouvelle table en dépliant la section Tables et sélectionnant l'option « New Table » qui apparait sur le clic droit :



4. Complétez la fenêtre de création de table comme ci-dessous.



5. Une fois la table créée, cliquez sur le bouton « New Query », dans la zone de texte, copiez la requête ci-dessous et terminez par cliquer sur le bouton Execute.

```
INSERT INTO Customers VALUES(1, 'Pierre', 'France');
INSERT INTO Customers VALUES(2, 'Bob', 'USA');
INSERT INTO Customers VALUES(3, 'Luc', 'France');
```

6. Grâce à l'utilitaire SQL Metal, il est possible d'automatiser la génération du DataContext correspondant à une base de données existante. Dans la barre de recherche du menu démarrer de votre ordinateur, recherchez « Visual Studio Command Prompt » et executez le. Placez vous dans le même répertoire que la base de données SQL CE que vous venez de créer.

7. Executeze la ligne de commande suivante, celle-ci va générer un data context nommé MonDataContext dans le fichier CustomersDataContext.cs en utilisant l'espace de nom MonApplication à partir de la base de données customers.sdf.

```
Sqlmetal /code :CustomersDataContext.cs /namespace :MonApplication
/context :MonDataContext customers.sdf
```

8. Reprenez le projet de l'exercice 6, et supprimez les classes **Customers**, **Country** et **MyDataContext**. Supprimez également le code dans le constructeur qui permettait la création et l'approvisionnement en données de la base.

9. Dans l'explorateur de solutions, faîtes un clic droit sur le projet et choisissez d'ajouter un élément existant et choisissez le fichier .cs généré par SQLMetal.
   Ce fichier n'est pas complètement commpatible avec Windows Phone 7, vous devez supprimez les deux constructeurs qui prennent en paramètre un objet implémentant l'interface **IDbConnection**.

10. Ajoutez la base de données .sdf au projet.

11. Modifiez la classe **MainPageViewModel** afin de faire appel au nouveau Data Context. Faîtes particulièrement attention à la chaine de connecion ! Prefixez la de « appdata ». Vous devez obligatoirement ouvrir la base en lecture seule.

```
public class MainPageViewModel
{
    private readonly MonDataContext _dataContext;

    private readonly ObservableCollection<Customers1> _customers
        = new ObservableCollection<Customers1>();

    public ObservableCollection<Customers1> Customers { get { return _custo
mers; } }

    public MainPageViewModel()
    {
        _dataContext = new MonDataContext("Data Source='appdata:customers.s
df';File Mode = read only;");

    }

    public void OnNavigatedTo()
    {
        new Thread(() =>
        {
            var customers = (from c in _dataContext.Customers1
                             where c.Country == "France"
                             select c).ToList();

            foreach (var c in customers)
            {
                Deployment.Current.Dispatcher.BeginInvoke((Action<Customers
1>)(customer =>
                    {
                        _customers.Add(customer);
                    }), c);
            }
        }).Start();
    }
}
```

Résumé

Dans cet atelier vous avez découvert comment récupérer des donnnées depuis une source située sur internet ou depuis un autre téléphone. Vous avez ensuite découvert comment stocker ces données dans l'isolated storage ou dans une base de données SQL CE. Enfin, vous avez découvert qu'il est possible de réutiliser une base de données SQL CE existante et de la déployer avec son application.