4/16/2008

MICROSOFT SWITZERLAND

SILVERLIGHT 2 BETA 1 HANDS-ON LAB

Based on the Comparis Silverlight Challenge | Sascha P. Corti

Hands-On Lab created and written by Sascha P. Corti
(sascha.corti@microsoft.com)

Sample code and lots of ideas contributed by Ronnie Saurenmann
(ronnie.saurenmann@microsoft.com)

Comparis Silverlight Challenge sample by Benedikt Unold
(benedikt.unold@comparis.ch)

Special thanks to the imperturbable testers:
Benedikt Unold
Marcel Trümpy
Olaf Feldkamp

Document Version 18 (Web) English

Special Annotations

Throughout the hands-on lab you will find the following annotations:

Goal: This explains the goal of the current chapter.

Note: Notes inform you of an important aspect of the programming steps you were just taking.

Hint: Hints may point out other possibilities to reach a goal.

Source code Snippets:

```
<Grid Height="120" HorizontalAlignment="Stretch" VerticalAlignment="Top"
x:Name="GridUserInput">
   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="0.5*"/>
      <ColumnDefinition Width="0.5*"/>
   </Grid.ColumnDefinitions>
</Grid>
```

Source Code Snippet 2a

- Each snippet has an associated number and can be found in the HOL Zip file in the file "**Source Code Snippets.txt**" under this number.

Contents

Introduction

This hands-on lab is based on **Silverlight 2 Beta 1** which was introduced at the Mix event in March 2008 in Las Vegas. It can't be guaranteed that all steps will work in later versions of Silverlight 2.

Security Warning

Some of the SQL Code demonstrated in this Hands-On lab is concatenated from user input and is therefore vulnerable to SQL injection attacks. We chose this format to keep the code in the lab short and discourage the use of this form of code for production use.

To learn more about SQL injection, please read the following article on MSDN:
http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/default.aspx

Prerequisites

Level: 200, Intermediate

- Knowledge about Visual Studio and .NET development (C#) recommended.
- Basic knowledge of XAML recommended.
- No Silverlight know-how required.

- Visual Studio 2008 Professional or above.
    - Download of the trial edition from: http://www.microsoft.com/downloads/details.aspx?familyid=83C3A1EC-ED72-4A79-8961-25635DB0192B&displaylang=en
- SQL Server Express (Part of the Visual Studio 2008 installation)

Installation

- Make sure that you have all previous alpha-versions of the Silverlight 1.1 uninstalled.
- Install the **Silverlight 2 Beta 1 runtime** from http://www.microsoft.com/silverlight/resources/install.aspx?v=2.0
    - Make sure that you have uninstalled the Silverlight 1.1 Alpha Tools for Visual Studio 2008 if you had installed them previously.

- Now install the Visual Studio 2008 Silverlight tools from http://www.microsoft.com/downloads/details.aspx?FamilyId=E0BAE58E-9C0B-4090-A1DB-F134D9F095FD&displaylang=en
    - BradleyB has a very good article on installing the Silverlight Tools Beta 1 for Visual Studio 2008 at http://weblogs.asp.net/bradleyb/archive/2008/03/06/installation-tips-for-sivliverlight-tools-beta-1-for-visual-studio-2008.aspx in case you run into trouble.
- Next, install Expression Blend 2.5 Beta from http://www.microsoft.com/expression/products/download.aspx?key=blend2dot5
    - This installation of Blend does not harm Expression Blend 1 or Expression Blend 2 previously installed on your system.
- Navigate to the "**`C:\Program Files\Microsoft SDKs\Silverlight\v2.0\Documentation\Help`**" folder. You will notice that "**`SilverlightDocumentation.chm`**" is only a dummy file. To get the real SDK documentation for Silverlight 2 Beta 1, extract the zip file downloadable from http://go.microsoft.com/fwlink/?LinkId=111131 to the SDK folder mentioned above.
- To get Visual Studio 2008 integration with the Silverlight documentation, follow these steps:
    - Open **Visual Studio 2008** as an **administrator**.
    - In the **Help** menu, choose **Index**.
        - Microsoft Document Explorer displays.
    - In the "**Filtered by"**: drop-down, choose (**unfiltered**).
    - In the **Look for** field, type "**Collection Manager"**.
    - Below the Collection Manager heading, double click **Help**.
    - Below the "**Collections available for inclusion in VSCC"** heading, check "**Microsoft Silverlight 2 SDK Documentation"**.
    - Click "**Update VSCC**".

Note: It will take only a few minutes to update the collection and merge in the Silverlight help files.
The best resource for up-to-date Silverlight tools is http://silverlight.net/GetStarted/

Preparing the Comparis Silverlight Challenge Solution

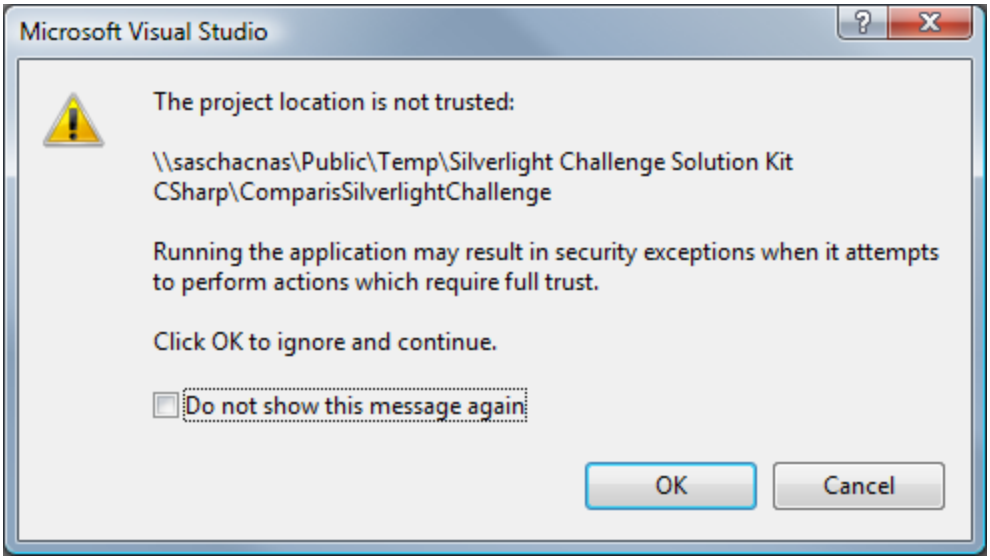
Goal: We will be basing this hands-on lab on the Comapris Silverlight Challenge (http://www.silverlightchallenge.ch). After completing this lab, you will have created a working Silverlight 2 Beta 1 application and have all the know-how you need to participate in the Comapris Silverlight Challenge with your own solution.

- Create a folder named "**`C:\Silverlight-HOL`**"
- Extract the contents from "**`Silverlight Challenge Solution Kit CSharp.zip`**" into the folder "**`C:\Silverlight-HOL`**"

- Using Visual Studio 2008, open the solution file that was extracted in "**`C:\Silverlight-HOL\Silverlight Challenge Solution Kit CSharp\ComparisSilverlightChallenge.sln`**"

Note: Some of the file names in the sample solution are rather long so extracting the ZIP file into an already deep folder hierarchy may result in extraction errors.

Note: Visual Studio may prompt you with a security dialog, mentioning that the project location is not trusted. Allow Visual Studio to access your solution. Furthermore, make sure that you have full read-/write-access to all the project files.

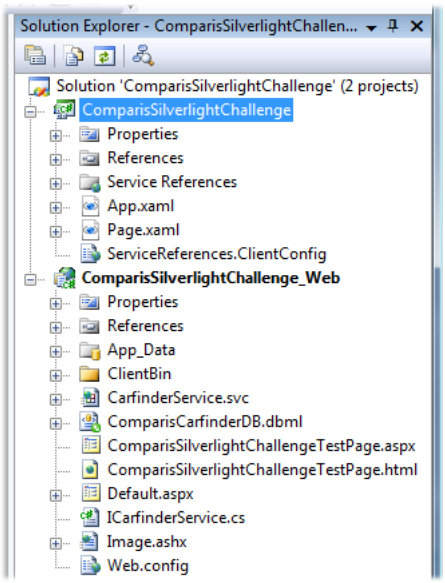


Hint: The sample solution from the Comparis Silverlight Challenge on top of which we will be building our application already has a few things prepared, including the entire project structure and the needed Web services. If at a later time you want to learn how to create a Silverlight project from scratch, you can look at the hands-on labs provided on http://silverlight.net/learn/labs.aspx

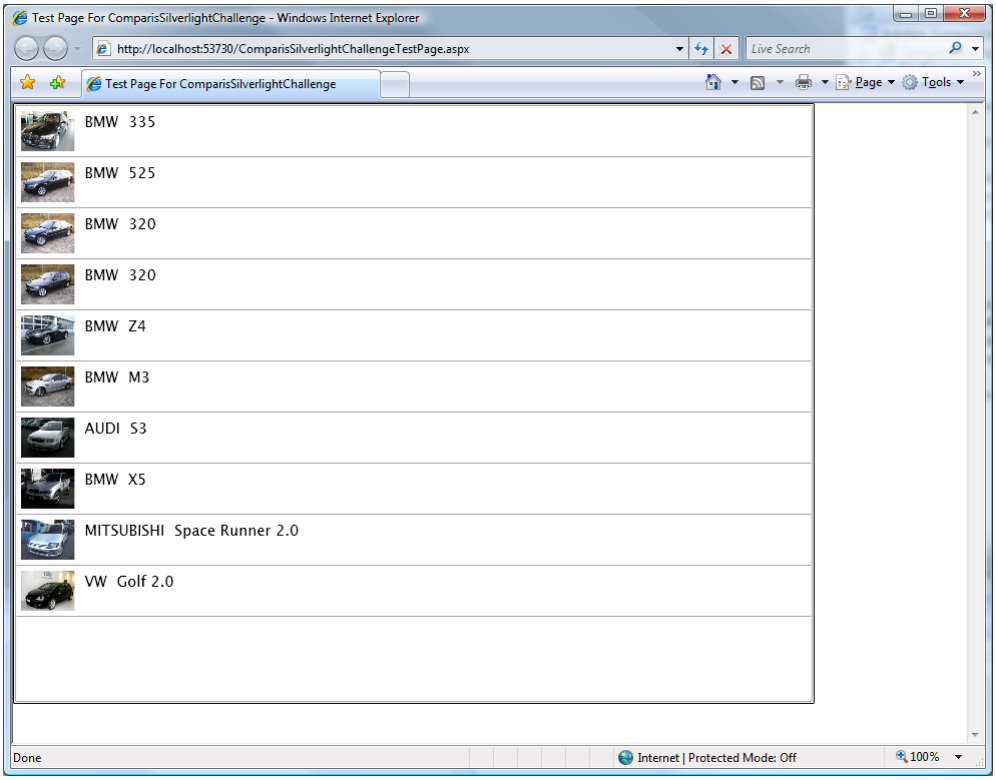
Exploring the Silverlight Challenge Sample Solution

Goal: Get to know the Silverlight Challenge sample solution provided by Comparis.

- First take a look at the solution in Visual Studio's **Solution Explorer**. You see two projects:
  - **ComparisSilverlightChallenge**, the main project in which the Silverlight control is created and
  - **ComparisSilverlightChallenge_Web**. Visual Studio automatically generates an associated web project if a new Silverlight project is created with the **_Web** extension.

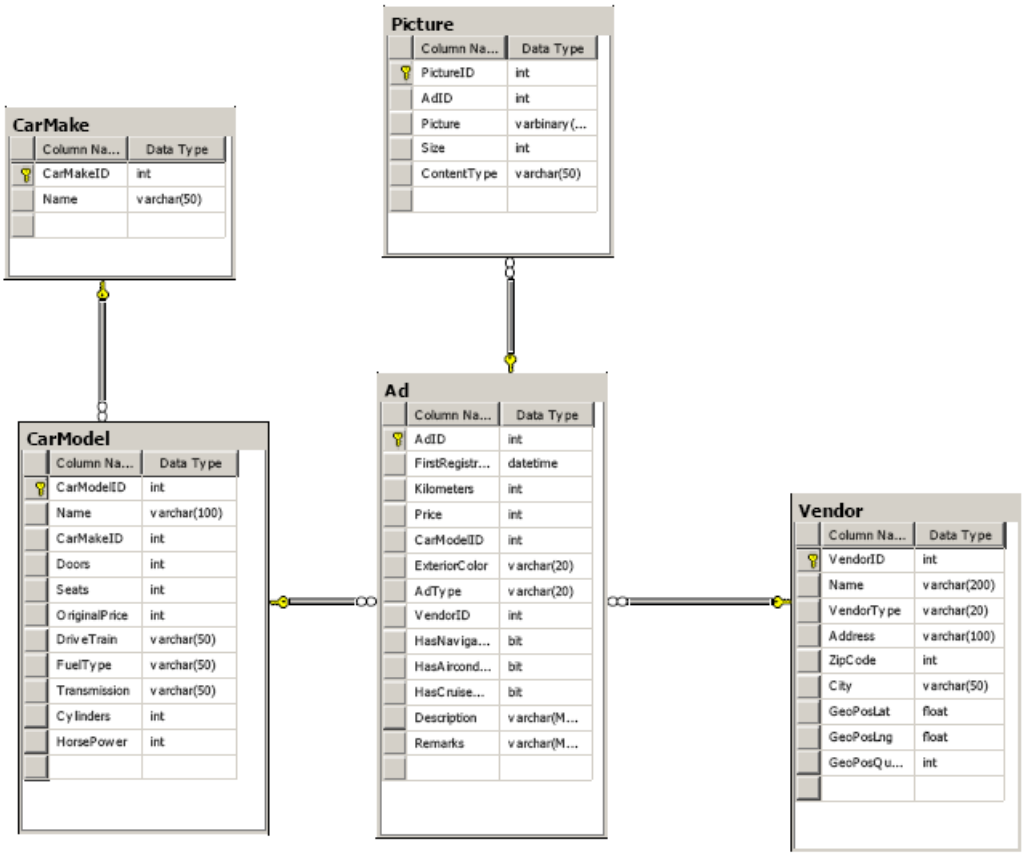


- Take a closer look at the second project in our solution, **ComparisSilverlightChallenge_Web**. This project contains three files by default:
  - **Default.aspx** which is empty
  - **{project name}TestPage.aspx** which is an ASPX page containing an **asp:Silverlight** control which embeds the main Silverlight control **Page.xaml** from the Silverlight project.
  - **{project name}TestPage.html** which is a static HTML page also containing a **div** which embeds the main Silverlight control **Page.xaml** from the Silverlight project.
- Open and inspect **ComparisSilverlightChallengeTestPage.aspx**. You will see the **asp:ScriptManager** control used for asynchronous Web service communication and the **asp:Silverlight** control which occupies the entire page (`Width="100%" Height="100%"`).
- Right click **ComparisSilverlightChallengeTestPage.aspx** and make it the start page of the project (Menu item "**Set as start page**").
- You can remove the generated files **ComparisSilverlightChallengeTestPage.html** and **Default.aspx**, we won't be needing them.
- Run the application by **hitting F5** in Visual Studio.



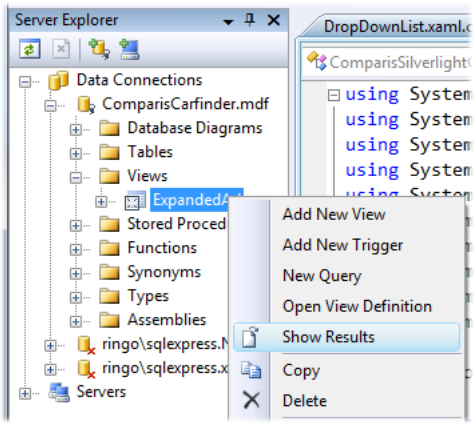
- This is the provided sample, a Silverlight control displaying a data grid populated with a few cars including a small image.
- Close the web browser.

- Let's now look at the data structure of the solution. Inspect **ComparisCarFinder.mdf** which can be found in the **ComparisSilverlightChallenge_Web** project's **App_Data** folder.



ComparisCarFinder.mdf data structure

- We are interested in the **View** called **ExpandedAd** that returns all information in a single table.
- Inspect the data by double-clicking on **ComparisCarFinder.mdf**, then opening the **Views** node in Server Explorer, right-clicking on **ExpandedAd** and selecting **Show Results.**



- Open **ComparisCarFinderDB.dbml**, the LINQ to SQL class used to access the database and **CarFinderService.svc**, the **Windows Communication Foundation** service already implemented that exposes the data as a web service.

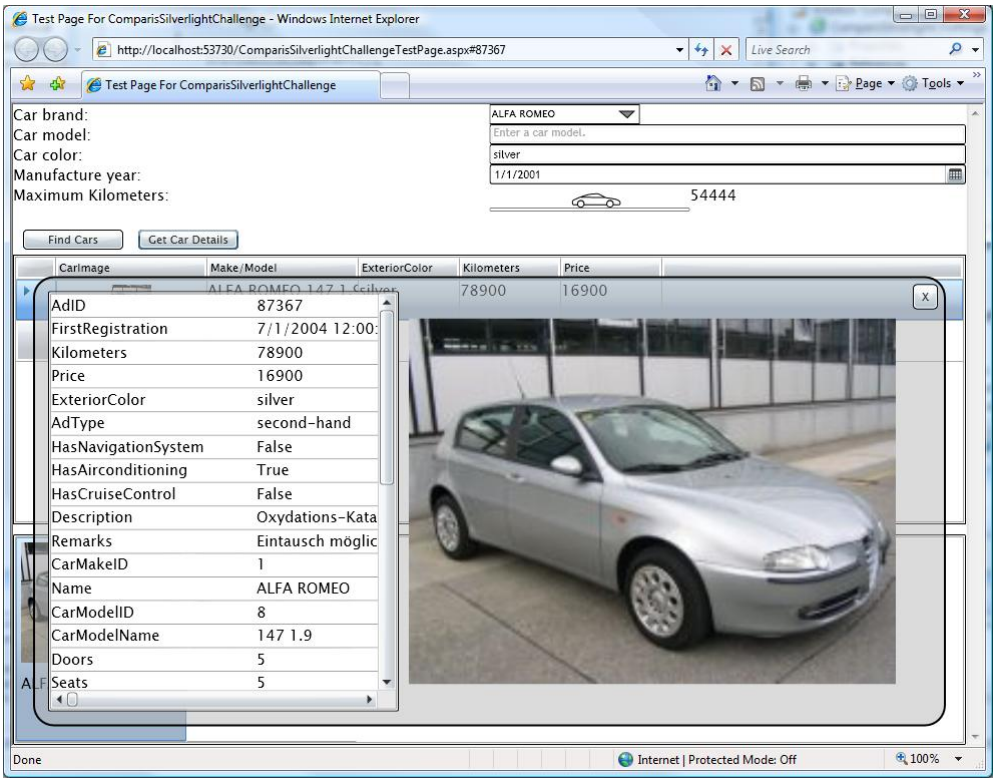
- Open **CarFinderService.svc.cs** and inspect the **ExpandedAd** – region to see the methods that we are going to access, **GetExpandedAd** and **GetExpandedAds**.
- Open **web.config** in the **ComparisSilverlightChallenge_Web** project and scroll to the bottom of the file.

Note: in the `<services>` -tag, the `<endpoint>` sets `binding="basicHttpBinding"`. This is currently the only binding that Silverlight can use to call a WCF service.

- In the Silverlight project **ComparisSilverlightChallenge**, the web service reference has been added for us as well. You will find it in the **Service References** folder as **CarfinderSvc** with its associated **ServiceReferences.ClientConfig**.
- Inspect **ServiceReferences.ClientConfig**. This configuration file knows where to find the services' endpoint.
- Close all open files in Visual Studio to make some room.

Scope of our own Silverlight Application

- We want to make our own application more flexible than the provided sample and allow the user to search for specific cars based on criteria like car brand, make, color, age and kilometers. We also want to improve the visual representation of the car advertisements.

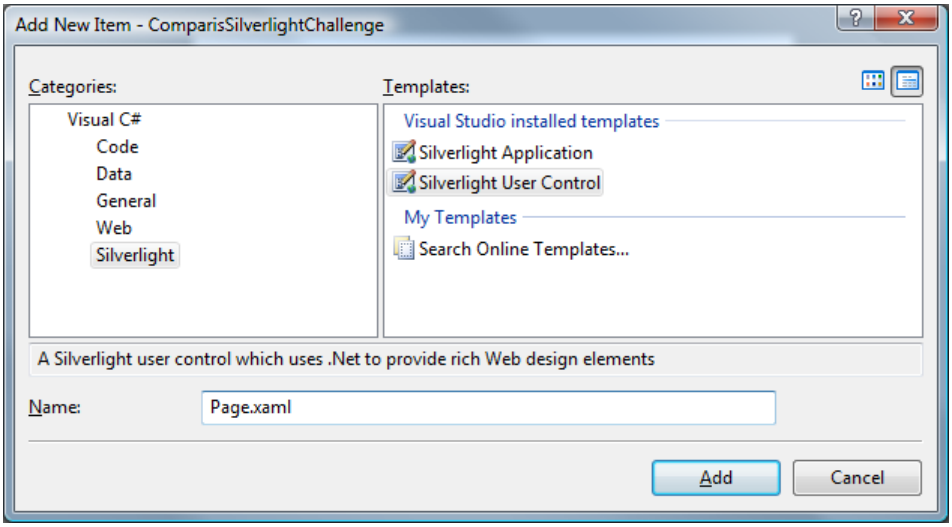


Preview of our final Carfinder Silverlight application.

Step 1: Creating our own Silverlight User Interface

Goal: Replace the existing sample application provided by the demo solution with a blank Silverlight user control.

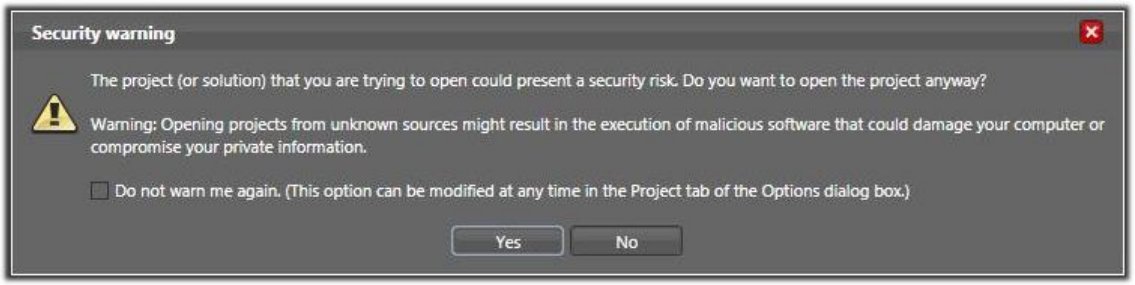
- In **Visual Studio**, delete the existing **`Page.xaml`** and **`Page.xaml.cs`** from the Silverlight Project **`ComparisSilverlightChallenge`**.
- Right-click the project, select "**Add new Item**", choose **Silverlight** in the categories list and click on "**Silverlight User Control**". Name the new control **`Page.xaml`**. Click the Add-button.



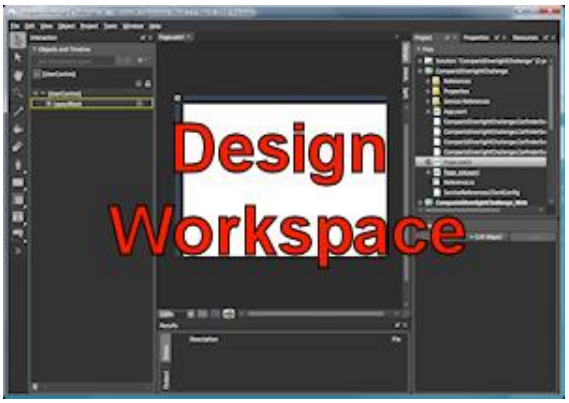
Hint: If you inspect **`App.xaml.cs`**, you will see that the Visual-Studio Add-In created an **Application_Startup** method that loads the **Page** class.

- Right-click the newly generated **`Page.xaml`** and select "**Open in Expression Blend**".
- In **Expression Blend**, select the menu **Window**, and choose **"Active Workspace**", "**Design Workspace**".

Note: Just like Visual Studio, Expression Blend may also prompt you with a security dialog, mentioning that the project location is not trusted. Allow Expressin Blend to access your solution.

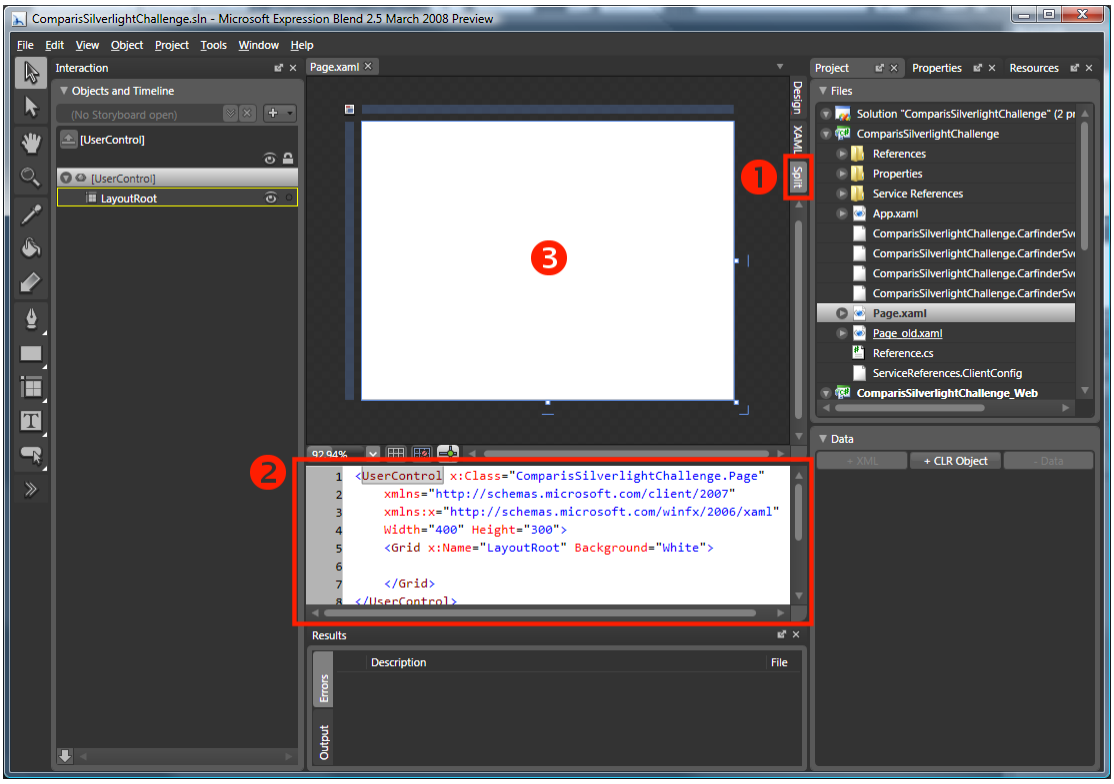


Hint: You can use **F6** to toggle between the Design- and Animation-Workspace.



- Select the **Split** view ❶ to show the source-code pane ❷ and the design view ❸
- Right-click the design-view and select "**Fit to screen**" in the **View**-menu.

Hint: You can do this whenever you want to automatically resize the design view to show all content by hitting CTRL+0.



Step 2: Adding Input Fields

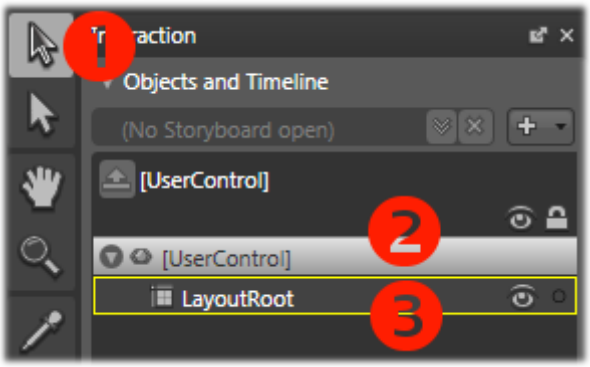
Goal: First we want to create input fields to let the user search for specific cars.

- First, we need to make sure that the **LayoutRoot** is **active** in Expression Blend.

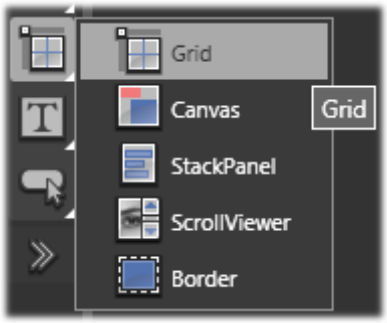
Note: Remember, that Expression Blend distinguishes between:

- **Selected control** ❷: The control that we are manipulating and that are represented in the properties pane. Selected controls appear with a light grey background.
- **Active control** ❸: The control that is the parent for the subsequent controls which will be added as child nodes. Active controls appear with a yellow border.

- Use the selection tool ❶ and single-click on a control in the "**Objects and Timeline**" window to select it, double-click to activate it.

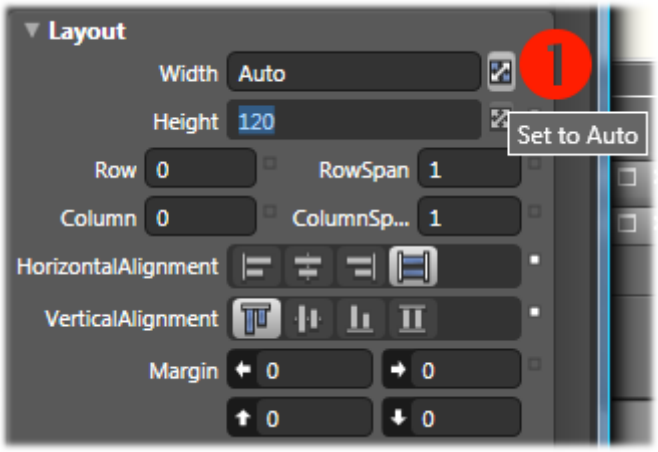


- Add a **Grid** control to **LayoutRoot**. To select the Grid control, click and hold the encapsulation controls button in the Expression Blend toolbox and select **Grid** from the choice of controls.



- "*Draw*" the **Grid** control in the top third of the canvas and set the layout-properties as follows.

Note: Remember that you can click the  -Button to set the width or height to "Auto". You may have to reveal the properties pane by clicking its tab if the project- or resources-tab is visible.



- Name the Grid control **GridUserInput** by right-clicking on the Grid control in the "Objects and Timeline" window and selecting **Rename**.

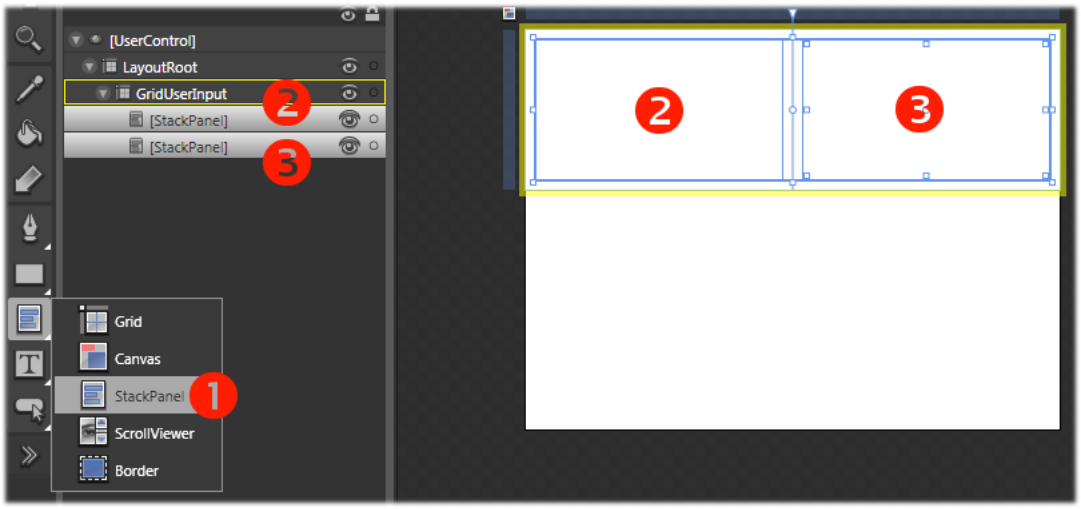
- Make the Grid control the active control (by double-clicking it) and place a **ColumnDefinition** inside the Grid by clicking on the blue bar above the Grid control.

- In the XAML source, set the **Width** element of both **ColumnDefinition** tags to **"0.5*"** which makes them proportional to the control width and the same size.

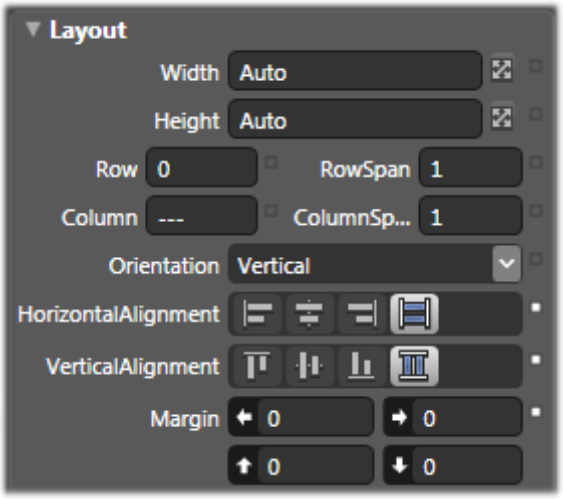
```
<Grid Height="120" HorizontalAlignment="Stretch" VerticalAlignment="Top"
x:Name="GridUserInput">
   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="0.5*"/>
      <ColumnDefinition Width="0.5*"/>
   </Grid.ColumnDefinitions>
</Grid>
```

Source Code Snippet 2a

- Now, select the Stack Panel control ❶ (again by clicking and holding the encapsulation controls in the toolbox) and draw a stack panel ❷ & ❸ in each column of the Grid control.

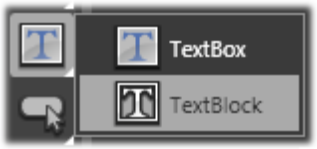


- Name the StackPanels **StackPanelLeft** and **StackPanelRight**.
- Now select both StackPanel controls in the "**Object and Timeline**" window and edit their layout as follows:



- The **Column** filed should be "**0**" for **StackPanelLeft** and "**1**" for **StackPanelRight**.
- Make **StackPanelLeft** the active control.
- Add a **TextBlock** control to **StackPanelLeft**. The TextBlock control can be found in the text controls in the Expression Blend toolbox.

Note the difference between **TextBox**, used for user input and **TextBlock**, used to display text.



- In the XAML source code, set the **TextBlock** properties as follows:

```
<StackPanel HorizontalAlignment="Stretch" Margin="0,0,0,0" VerticalAlignment="Stretch"
x:Name="StackPanelLeft">
   <TextBlock Height="20" Width="Auto" Text="TextBlock" TextWrapping="Wrap"/>
</StackPanel>
```

Source Code Snippet 2b

- Copy the **TextBlock** control 4 times and edit only the "**Text=**" attributes. The StackPanel that encloses the TextBlocks will take care of the proper positioning, "stacking" of the TextBlocks.

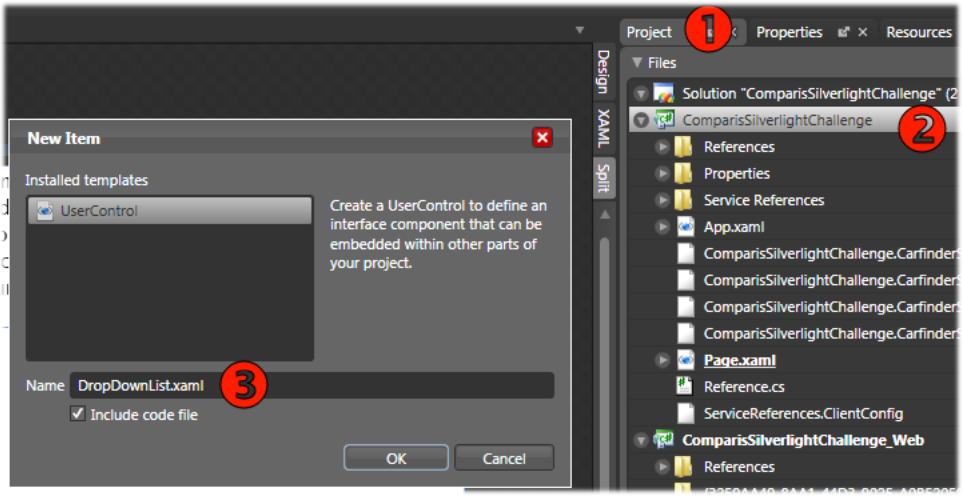
```
<StackPanel HorizontalAlignment="Stretch" Margin="0,0,0,0" VerticalAlignment="Stretch"
x:Name="StackPanelLeft">
   <TextBlock Height="20" Width="Auto" Text="Car brand:" TextWrapping="Wrap"/>
   <TextBlock Height="20" Width="Auto" Text="Car model:" TextWrapping="Wrap"/>
   <TextBlock Height="20" Width="Auto" Text="Car color:" TextWrapping="Wrap"/>
   <TextBlock Height="20" Width="Auto" Text="Manufacture year:" TextWrapping="Wrap"/>
   <TextBlock Height="20" Width="Auto" Text="Maximum Kilometers:" TextWrapping="Wrap"/>
</StackPanel>
```

Source Code Snippet 2c

Step 3: Building a Custom Control

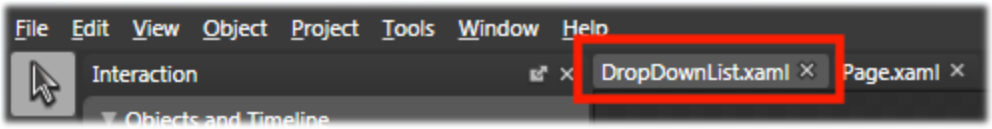
Goal: For the car brand, we want to present the user with a pre-populated **drop-down list** of all available brands. Silverlight 2 Beta 1 does not have a drop-down list, so we will create a **user control** with this functionality.

- Select the "**Project**" tab ❶, right-click the "**ComparisSilverlightChallenge**" project and select "**Add New Item…**" ❷. Name the new UserControl "**DropDownList.xaml**" ❸, make sure the "**Include code file**" checkbox is selected and confirm by clicking **OK**.



- Make sure that **DropDownList.xaml** is open in the editor.

Note: Remember that you can use the tabs to switch between open documents.

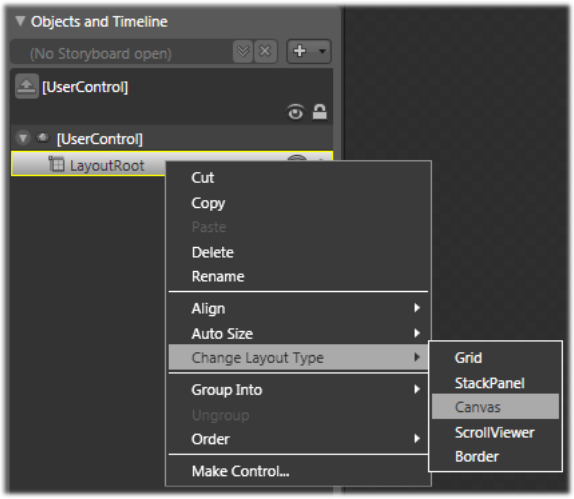


- In the XAML source window, change the **DesignWidth** and **DesignHeight** attributes of the **UserControl** tag as follows. This will render our control properly in the Expression Blend design view.

```
<UserControl
   xmlns="http://schemas.microsoft.com/client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   x:Class="ComparisSilverlightChallenge.DropDownList"
   d:DesignWidth="150" d:DesignHeight="250">
```

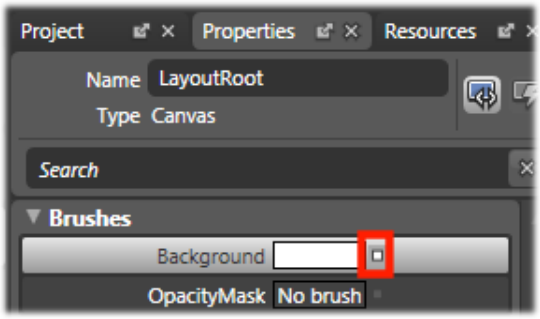
Source Code Snippet 3a

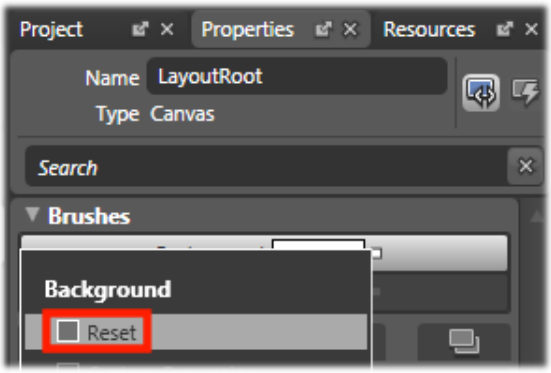
- Right-click the **LayoutRoot** control in the **Objects and Timeline** window and select "**Change Layout Type**" "**Canvas**".



- With **LayoutRoot** selected, click the **little square** right to the "**Brushes**", "**Background**" property in the **properties window** and select "**Reset**".

Note: Remember that with this option, we can reset any customization of any control in Expression Blend to its default setting.

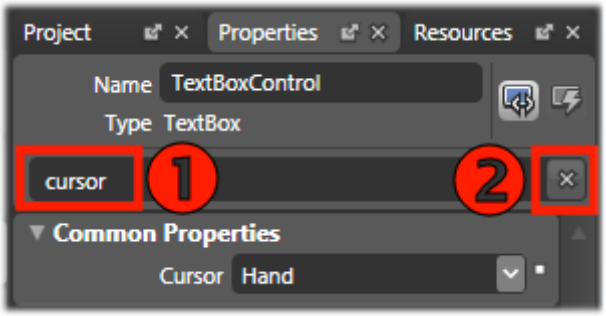




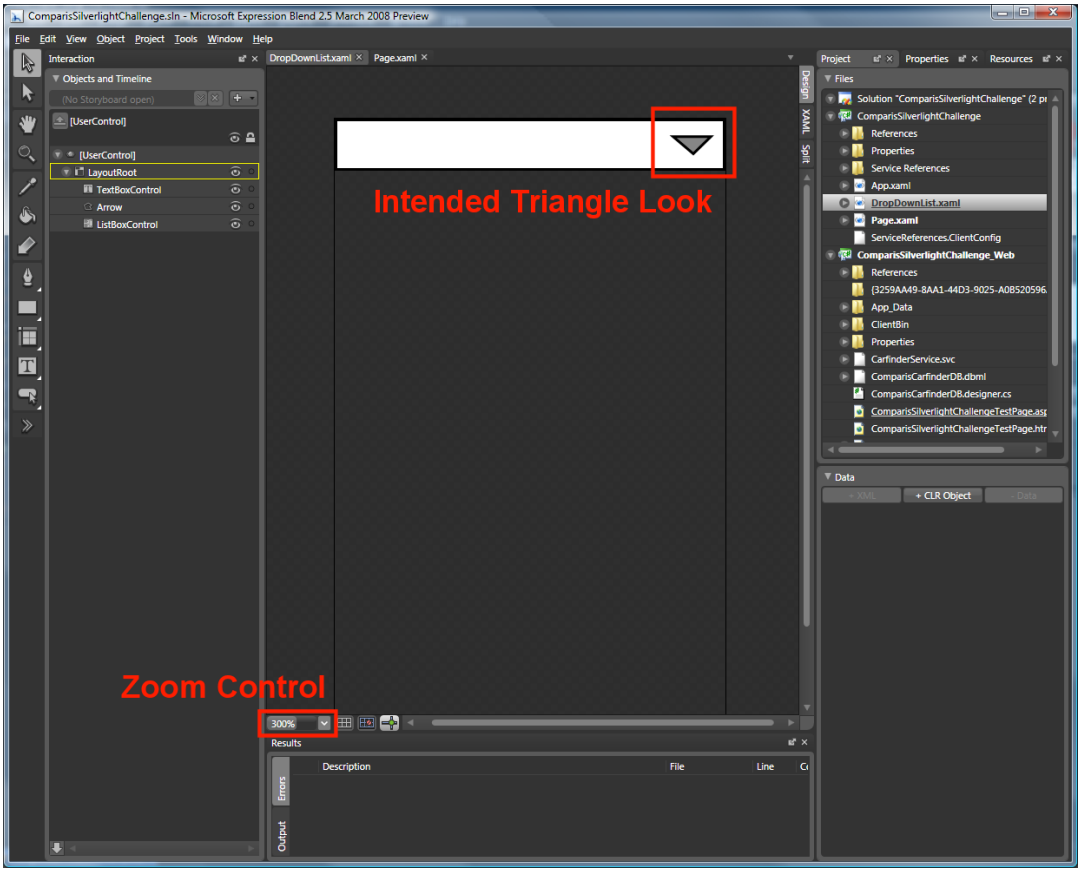
- Add a **TextBox** control to **LayoutRoot** (make sure that LayoutRoot is active) and set the following properties:

Property	Value
Name	TextBoxControl
Width	150
Height	20
Text	{Reset to empty}
FontSize	10
IsReadOnly	True
Cursor	Hand

Hint: If you are unable to find a property in Expression Blend's properties window, you can search for it using the **Search-field** ❶.
Also remember to clear the search after you use it to see all the properties again❷ .
You can also edit the properties in the **XAML source window**, just make sure that Expression Blend accepts your manual changes and doesn't display an error. It is a good practice to **build** (CTRL+B) the project after doing manual changes to see if everything compiles OK.

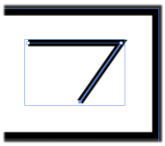
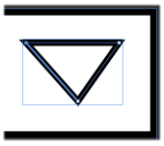


- We will use a **Path** to draw the triangle that the user will click to open the drop-down box.
- First, use the **zoom control** in Expression Blend to zoom the workspace so that the **TextBoxControl** fills most of your work area.



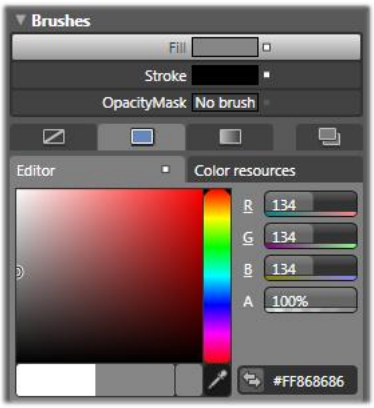
Zoom Control close-up

- Now select the **Pen** tool and draw the triangle by single-clicking each of the corners of the triangle.

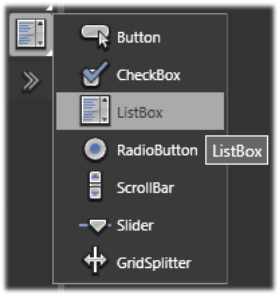
 Click 1	 Click 2	 Click 3	 Click 4

- Make sure that the last click closes the shape.
- Now use the "**Direct Select**" tool from the tool box and drag the corners and sides of your Path until you like the resulting triangle.

- Finally, set the "**Brushes**" "**Fill**" property of the Path to a medium grey and name the Path "**Arrow**".



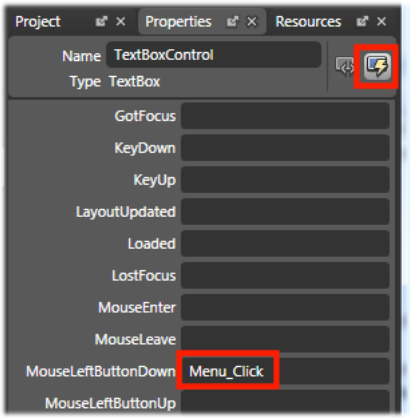
- Place a **ListBox** underneath the **TextBoxControl** on the **LayoutRoot**. The ListBox can be found in the last control section in the toolbox.



- Set the following properties for the ListBox control:

Property	Value
Name	ListBoxControl
Width	150
Height	150
Canvas.Top	20
Visibility	Collapsed

- Let's add the **event handlers** needed for the dropdown box to work properly. Select (=single click) the **TextBoxControl** and click the **Events-button** in Expression Blend's Properties-window.
- In the **MouseLeftButtonDown** event field type **Menu_Click**.



Note: This automatically generates the event handler method in the **DropDownList** class in **DropDownList.xaml.cs** and the event handler on the Control in **DropDownList.xaml**.

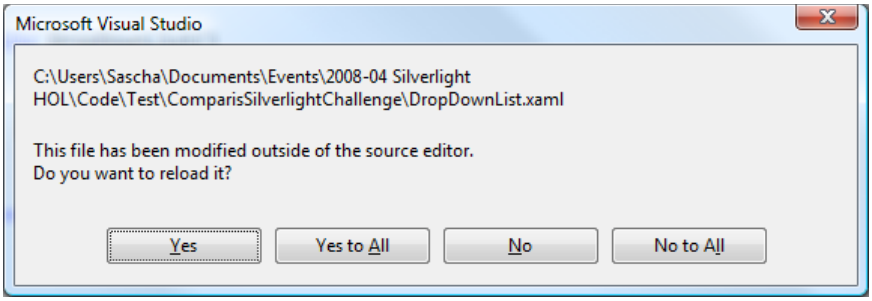
Note: If Expression Blend stops with an error, saying that the code-behind file could not be changed, switch to Visual Studio and refresh the files and rebuild the solution. Then switch back to Expression Blend and try again.

- Use this method to add the following, additional events to the controls:

Control	Event	Handler
Arrow	MouseLeftButtonDown	Menu_Click
ListBoxControl	SelectionChanged	ListBoxControl_SelectionChanged

- Save your project and switch over to **Visual Studio 2008**.

Hint: Notice that whenever you edit and save the project in one of the tools, the other tool will prompt you if it should reload the data to reflect the latest changes. Always respond with "**Yes to all**".



- Edit "**DropDownList.xaml.cs**" and add the **Menu_Click** event logic. This will toggle visibility of the ListControl.

```
private void Menu_Click(object sender, MouseButtonEventArgs e)
{
   ListBoxControl.Visibility = (ListBoxControl.Visibility == Visibility.Visible)
      ? Visibility.Collapsed : Visibility.Visible;
}
```

Source Code Snippet 3b

- Now add the **ListBoxControl_SelectionChanged** event handler.

```
private void ListBoxControl_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
   Type itemType = Type.GetType(ListBoxControl.SelectedItem.GetType().Name);

   if (ListBoxControl.SelectedItem.GetType().GetProperty(
      ListBoxControl.DisplayMemberPath) != null)
      TextBoxControl.Text = ListBoxControl.SelectedItem.GetType().GetProperty(
         ListBoxControl.DisplayMemberPath).GetValue(
         ListBoxControl.SelectedItem, null).ToString();
   ListBoxControl.Visibility = Visibility.Collapsed;
}
```

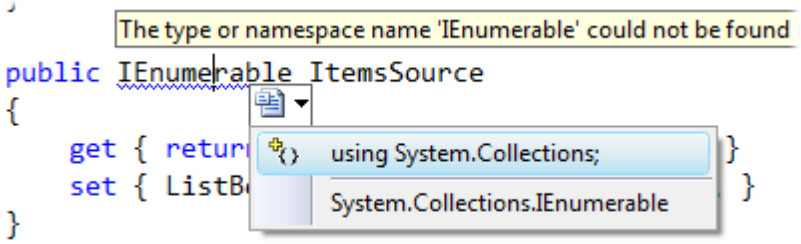
Source Code Snippet 3c

- Add the **public accessors** for **ItemsSource**, **DisplayMemberPath** and **SelectText**:

```
public IEnumerable ItemsSource
{
   get { return ListBoxControl.ItemsSource; }
   set { ListBoxControl.ItemsSource = value; }
}

public string DisplayMemberPath
{
   set { ListBoxControl.DisplayMemberPath = value; }
   get { return ListBoxControl.DisplayMemberPath; }
}

public string SelectText
{
   get { return TextBoxControl.Text; }
}
```

Source Code Snippet 3d

Hint: When typing "**IEnumerable**", Visual Studio 2008 will tell you that it needs a reference to **System.Collections** namespace. Use the **tooltip** to automatically have Visual Studio add the corresponding "using" statement.



- Make sure the solution **compiles** without errors.

Step 4: Adding the Custom Control to the Main Application

Goal: This is all we need for our ListBox control to work. Now, let's add it to our main Silverlight control.

- Switch to **Expression Blend**.
- Activate **Page.xaml** and click the **Asset Library** button in the toolbox. Choose **Custom Controls** in the Asset Library and select our **DropDownList** control.

Hint: If **DropDownList** is not shown in "**Custom Controls**", you may have to rebuild the solution in Expression Blend (hit CTRL+B).

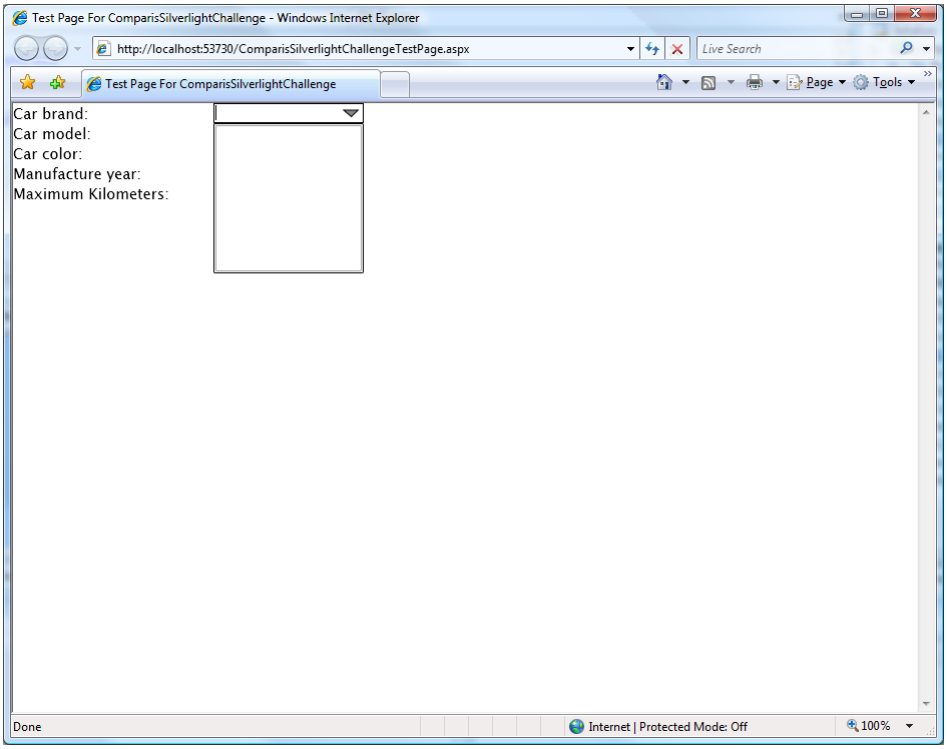
- Make **StackPanelRight** the active control and add a **DropDownList** control.
- Set the following properties:

Property	Value
Name	DropDownListCarMake
Width	Auto
Height	20
HorizontalAlignment	Left
VerticalAlignment	Top

Step 5: Running our Application for the First Time

Goal: Before we wire up the DropDown list, let's run our solution for the first time.

- **Save your work** in Expression Blend and switch over to **Visual Studio 2008**.
- Run the solution by hitting **F5**. You should see the functional Silverlight control with our DropDown list that is still empty.



Step 6: Populating the DropDown list

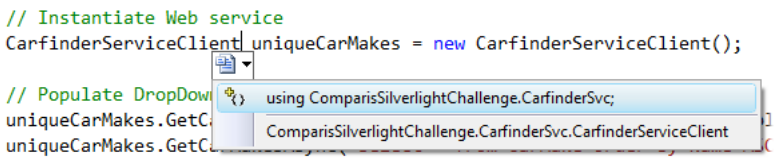
- Close the web browser and switch back to **Visual Studio**.
- We now need to link our Silverlight control to the WCF Service "**CarfinderService.svc**" supplied with the Comparis sample solution.
- Edit **Page.xaml.cs**. We will instantiate the web service when the control loads and populate the **DropDownListCarMake** with an alphabetically sorted list of unique car makes.
- Update the **Page** class constructor:

```
public Page()
{
   InitializeComponent();

   // Instantiate Web service
   CarfinderServiceClient uniqueCarMakes = new CarfinderServiceClient();

   // Populate DropDownBox control
   uniqueCarMakes.GetCarMakesCompleted += new
      EventHandler<GetCarMakesCompletedEventArgs>(uniqueCarMakes_GetCarMakesCompleted);
```

```
   uniqueCarMakes.GetCarMakesAsync("select * from CarMake order by Name ASC");
}
```

Source Code Snippet 6a

- When typing **CarFinderServiceClient**, you will again be prompted to add the **using** statement for the **ComparisSilverlightChallenge.CarfinderSvc** reference.



- Add the asynchronous event handler **GetCarMakesCompletedEventArgs** to the **Page** class:

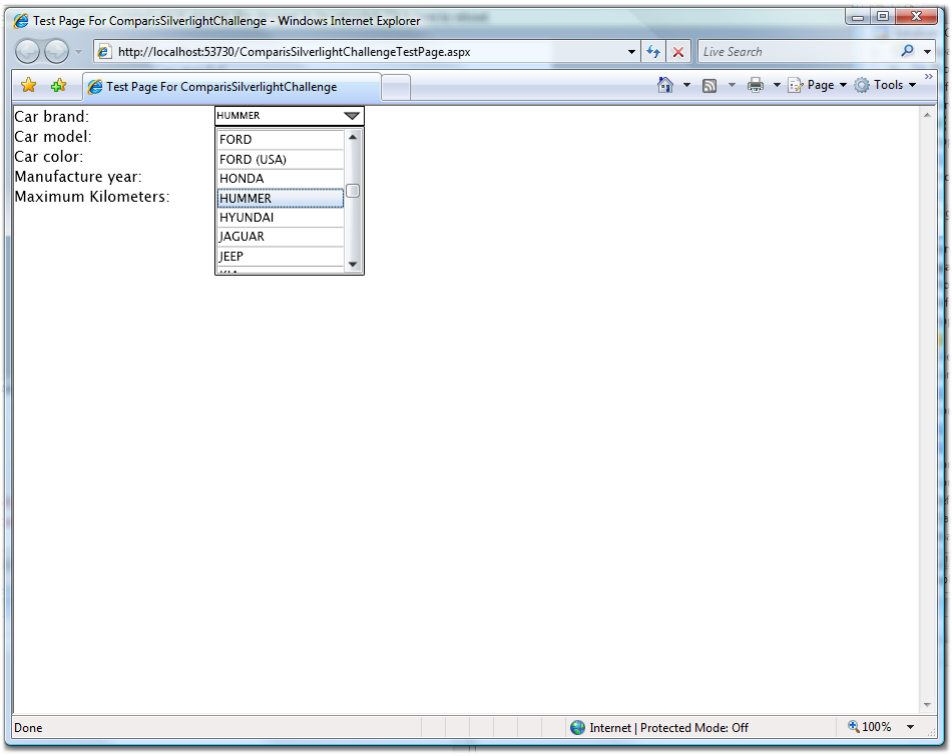
```
// Async event handler: DropDown Box population
void uniqueCarMakes_GetCarMakesCompleted(object sender, GetCarMakesCompletedEventArgs e)
{
   DropDownListCarMake.ItemsSource = e.Result;
}
```

Source Code Snippet 6b

- Now we need to bind the proper data field to the **DisplayMemberPath** property of the **DropDownListCarMake** control.
- Switch to **Expression Blend**, select the **DropDownListCarMake** control and search for the property called **DisplayMemberPath**.
- Enter "**Name**".
- We also need to make sure that the drop-down box is always displayed in front of all other controls when it is opened. To do this, we will set `Canvas.ZIndex="9999"`.

Note: We will have to do this change in the **XAML source** window, as Expression Blend does not currently allow us to set this property in the properties window.

- Save your work in **Expression Blend** and switch back to **Visual Studio.**
- Run the project (Hit F5) and test that the drop-down list shows its proper behavior. It should list all car makes alphabetically and allow you to select one.

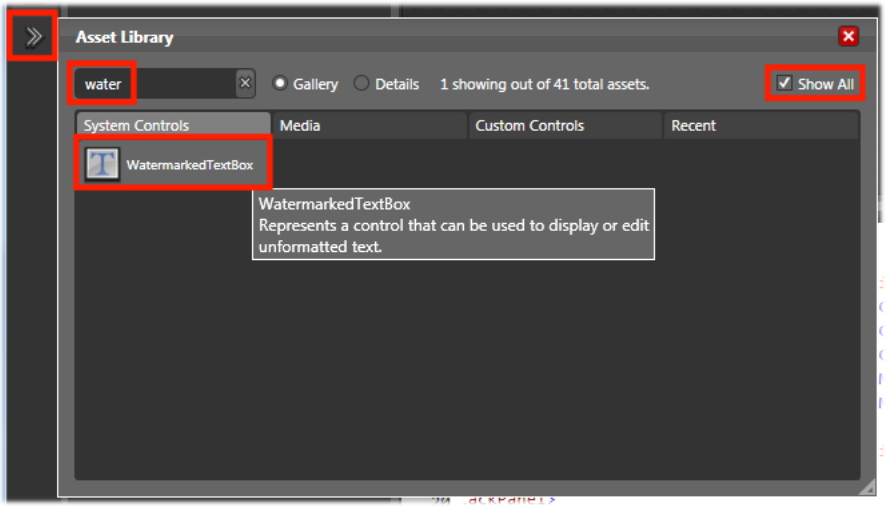


- Close the web browser.

Step 7: Completing the User Input Section

Goal: We will now add the remaining controls to the **GridUserInput** section.

- Switch to **Expression Blend** and open the **Asset Library**. Select "**Show all**" controls and search for the term "Water" to find the **WatermarkedTextBox**.



- Add a **WatermarkedTextBox** to **StackPanelRight** and set the following properties.

Property	Value
Name	TextBoxModel
Width	Auto
Height	20
Text	{reset to empty}
Watermark	Enter a car model.

- Add a second **WatermarkedTextBox** to **StackPanelRight** with the following properties.

Property	Value
Name	TextBoxColor
Width	Auto
Height	20
Text	{reset to empty}
Watermark	Enter a car color.

- Add a **DatePicker** to **StackPanelRight** (you will find the DatePicker control in the Asset Library).

Property	Value
Name	DatePickerYear
Width	Auto
Height	20

- Below **DatePickerYear**, add a **StackPanel** as follows.

Property	Value
Name	StackPanelSlider
Width	Auto
Height	20
Orientation	Horizontal

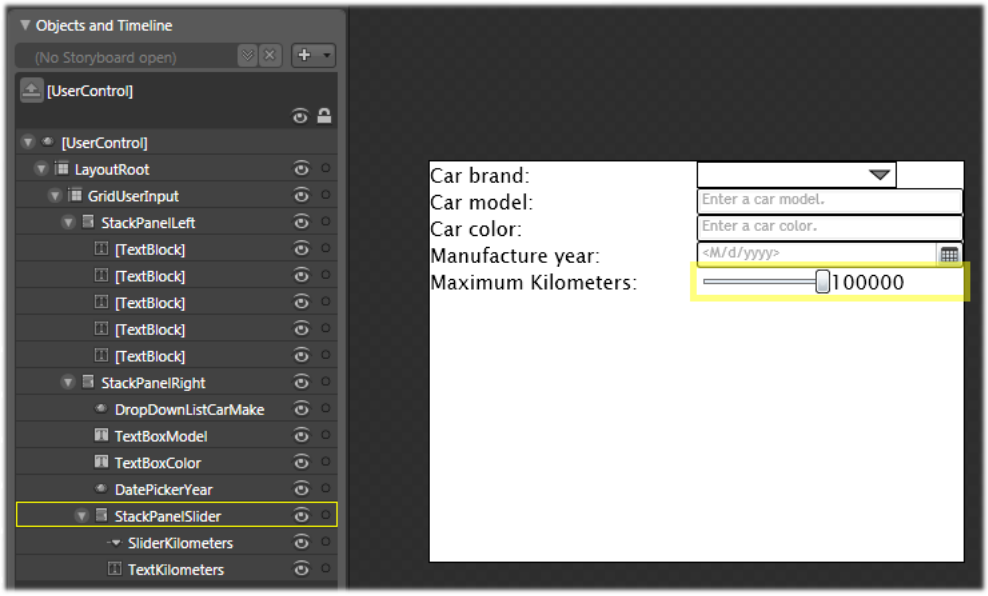
- Activate the **StackPanelSlider** control and add a **Slider**.

Property	Value
Name	SliderKilometers
Width	100
Height	20
LargeChange	10000
Maximum	100000
Minimum	100
SmallChange	1000
Value	100000

- Add a **TextBlock** to the **StackPanelSlider**:

Property	Value
Name	TextKilometers
Text	100000

- That concludes our **GridUserInput** section for now, your project should look as follows in Expression Blend.



- Activate **LayoutRoot** and add a **Button** control below the **GridUserInput**.

Property	Value
Name	ButtonCars
Width	100
Height	20
HorizontalAlignment	Left
VerticalAlignment	Top
Margin	10,125,0,0
Width	100
Content	Find Cars

Hint: To edit the **Margin** property, use the **margin editor** in the Layout-section of the Properties window..

Note: The order of values in XAML is: `Margin="[left], [top], [right], [bottom]"`.

- We are running out of space in the designer. Remember that we set the **asp:Silverlight** control in **ComparisSilverlightChallengeTestPage.aspx** to be 100% wide and 100% high. We want our Silverlight control to use all the space that it is given by the hosting asp:Silverlight server-side control. Look at the top `<UserControl` tag in Expression Blend. You will see that we have predefined `Width="400" Height="300"` that is currently limiting our workspace. **Remove them**.
- Problem: Now the control will be properly rendered in the browser, but in Expression Blend, the whole Silverlight control is resized to a minimal size which makes it hard to work on. For this case there is a set of special attributes, `DesignWidth` and `DesignHeight` that is Expression Blend specific and can be set to `Ignorable` for the running Silverlight application. To use these attributes, we need to import two namespaces,
  `xmlns:d="http://schemas.microsoft.com/expression/blend/2008"`
  `xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"`

- Update the top **UserControl** as follows.

```
<UserControl x:Class="ComparisSilverlightChallenge.Page"
   xmlns="http://schemas.microsoft.com/client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:ComparisSilverlightChallenge="clr-namespace:ComparisSilverlightChallenge"
   d:DesignWidth="800" d:DesignHeight="600"
   mc:Ignorable="d">
```

Source Code Snippet 7a

- Now Expression Blend will show the whole control in 800*600 pixels and the running application will ignore this fixed setting.

- Activate the **LayoutRoot** and add a **DataGrid** control as follows:

Property	Value
Name	DataGridCars
Margin	0,150,0,220
AutoGenerateColumns	True

Note: Expression Blend automatically added a new namespace for the DataGrid control: `xmlns:System_Windows_Controls="clr-namespace:System.Windows.Controls; assembly=System.Windows.Controls.Data"`.

Step 8: Querying the Data based on the User's Selections

Goal: Now we will wire up the Button-click event for **ButtonCars** and make it populate the data grid based on the user's selections.

- Save your work and switch to **Visual Studio**. This time, we will use Visual Studio to create the event handler declaratively and not set it Expression Blend.

- Make **Page.xaml.cs** the active file in Visual Studio's code editor.
- Build the project to make sure Visual Studio knows all of the newly added controls.
- In the **Page()** constructor of the Page class, after **InitializeComponent()** type "**ButtonCars.Click +=**". Visual Studio will now prompt you to automatically generate the routed event handler. Accept by pressing the **Tab key** twice.

```
ButtonCars.Click +=
                    new RoutedEventHandler(ButtonCars_Click);    (Press TAB to insert)
```

Press "Tab"

```
ButtonCars.Click +=new RoutedEventHandler(ButtonCars_Click);
                                          Press TAB to generate handler 'ButtonCars_Click' in this class
```

Press "Tab"

- You should get the following code:

```
public Page()
{
   InitializeComponent();

   // Declare ButtonCars Click event handler
   ButtonCars.Click += new RoutedEventHandler(ButtonCars_Click);
...
```

```
void ButtonCars_Click(object sender, RoutedEventArgs e)
{
   throw new NotImplementedException();
}
```

- Now replace the **ButtonCars_Click** method with your own code, generating a SQL query string to narrow the search results based on the user input.

```
// Event handler: ButtonCars_Click
void ButtonCars_Click(object sender, RoutedEventArgs e)
{
   StringBuilder sqlQuery = new StringBuilder();
   sqlQuery.AppendFormat(
      "select top 50 * from ExpandedAd where Name like '{0}%' and CarModelName like '{1}%'
and ExteriorColor like '{2}%'",
      DropDownListCarMake.SelectText, TextBoxModel.Text, TextBoxColor.Text);

   if (DatePickerYear.SelectedDate != null)
      sqlQuery.AppendFormat(" and FirstRegistration > '{0}'",
         DatePickerYear.SelectedDate);

   this.query_Cars(sqlQuery.ToString());
}
```

Source Code Snippet 8a

- **StringBuilder** will need a reference to the **System.Text** namespace.
- Now we need to implement the **query_Cars** method that calls the Web service based on the SQL query it is passed.

```
// Query web service for cars based on sqlQuery
```

```
private void query_Cars(string sqlQuery)
{
   CarfinderServiceClient client = new CarfinderServiceClient();

   System.ServiceModel.BasicHttpBinding binding =
      new System.ServiceModel.BasicHttpBinding();
   binding.MaxBufferSize = 2147483647;
   binding.MaxReceivedMessageSize = 2147483647;
   client.Endpoint.Binding = binding;

   client.GetExpandedAdsCompleted += new EventHandler<
      GetExpandedAdsCompletedEventArgs>(client_GetExpandedAdsCompleted);
   client.GetExpandedAdsAsync(sqlQuery.ToString());
}
```

Source Code Snippet 8b

Note: Here we set the **binding.MaxBufferSize** and **binding.MaxReceivedMessageSize** again. This has already been set in the **ServiceReference.ClientConfig**, but for now, Silverlight 2 Beta 1 ignores this setting.

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_ICarfinderService"
                 maxBufferSize="2147483647"
                 maxReceivedMessageSize="2147483647">
...
```

- Implement the (or edit the auto generated) event handler **client_GetExpandedAdsCompleted** to data bind the results from the Web service call to our **DataGridCars**.

```
// Async event handler: Web service call
void client_GetExpandedAdsCompleted(object sender, GetExpandedAdsCompletedEventArgs e)
{
   //Bug: DataGrid crashs when passed collection is empty!
   if (e.Result.Count() > 0)
   {
      myAds = e.Result.ToList();
      DataGridCars.ItemsSource = myAds;
   }
   else
   {
      DataGridCars.ItemsSource = null;
      HtmlPage.Window.Alert("No cars matching your qurey where found.");
   }
}
```

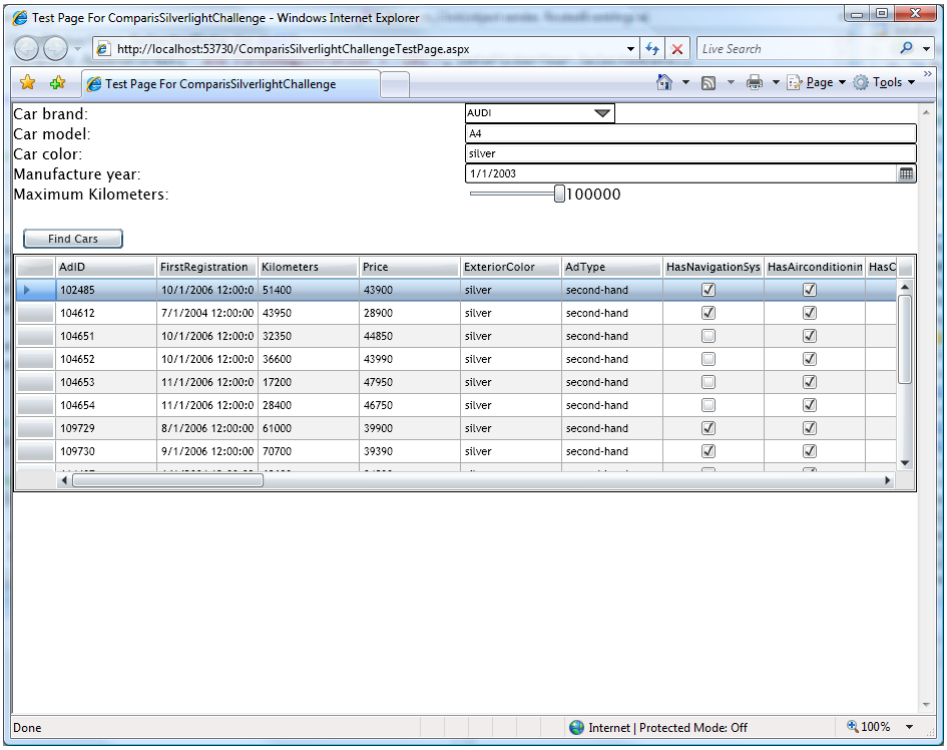
Source Code Snippet 8c

- **HtmlPage** needs a reference to **System.Windows.Browser**.
- Also, we need to declare the class-variable **myAds** as a list of type ExpandedAd.

```
public partial class Page : UserControl
{
   private List<ExpandedAd> myAds;
...
```

Source Code Snippet 8d

- Save your work, build the project and run it. Try a few car searches to see if your Silverlight app works as expected. We have not yet wired up the **SliderKilometers** slider control, so that's expected not to work at this time.



- Close the web browser.

Step 9: Adding a Car Photo Gallery

Goal: Now let's add another control that displays the images of the car our query returned in a horizontal gallery below the **DataGridCars**.

- Switch to **Expression Blend** and make **LayoutRoot** active and add a **ListBox** control beneath **DataGridCars**.

Property	Value
Name	ListBoxImages
Width	Auto
Height	210
HorizontalAlignment	Stretch
VerticalAlignment	Bottom
Margin	0,0,0,0

- Now we need to template our **ListBoxImages** because we want the images to be shown horizontally together with some text inside a StackPanel. We need to do this manually so we switch over to **Visual Studio** to get help from IntelliSense.
- Change **ListBoxImages** to the following

```
<ListBox Height="210" VerticalAlignment="Bottom" HorizontalAlignment="Stretch"
   Margin="0,0,0,0" x:Name="ListBoxImages">

   <ListBox.ItemsPanel>
      <ItemsPanelTemplate>
         <StackPanel Orientation="Horizontal"/>
      </ItemsPanelTemplate>
   </ListBox.ItemsPanel>

   <ListBox.ItemTemplate>
      <DataTemplate>
         <StackPanel>
            <Image Width="160" Height="120" Margin="5,5,5,5" />
            <StackPanel Margin="5,5,5,5" Orientation="Horizontal">
               <TextBlock Text="" />
               <TextBlock Margin="5,0,0,0" Text="" />
            </StackPanel>
         </StackPanel>
      </DataTemplate>
   </ListBox.ItemTemplate>

</ListBox>
```

Source Code Snippet 9a

- Let's populate the **ListBoxImages** control when the search button is pressed. Update the **client_GetExpandedAdsCompleted** event handler to do this:

```
// Async event handler: Web service call
void client_GetExpandedAdsCompleted(object sender, GetExpandedAdsCompletedEventArgs e)
{
   //Bug: DataGrid crashs when passed collection is empty!
   if (e.Result.Count() > 0)
   {
      myAds = e.Result.ToList();
      DataGridCars.ItemsSource = myAds;
      ListBoxImages.ItemsSource = myAds;
   }
   else
   {
      DataGridCars.ItemsSource = null;
      ListBoxImages.ItemsSource = null;
      HtmlPage.Window.Alert("No cars matching your qurey where found.");
   }
}
```

Source Code Snippet 9b

- We need to bind the data fields from **myAds** to the controls inside the **ListBoxImages**. Update the list box as follows.

```
<ListBox Height="210" VerticalAlignment="Bottom" HorizontalAlignment="Stretch"
Margin="0,0,0,0" x:Name="ListBoxImages">

   <ListBox.ItemsPanel>
      <ItemsPanelTemplate>
         <StackPanel Orientation="Horizontal"/>
      </ItemsPanelTemplate>
   </ListBox.ItemsPanel>

   <ListBox.ItemTemplate>
      <DataTemplate>
```

```
        <StackPanel>
          <Image Width="160" Height="120" Margin="5,5,5,5"
                 Source="{Binding PictureID,
                 Converter={StaticResource urlBuilderConverter}}"/>
          <StackPanel Margin="5,5,5,5" Orientation="Horizontal">
            <TextBlock Text="{Binding Name}" />
            <TextBlock Margin="5,0,0,0" Text="{Binding CarModelName}" />
          </StackPanel>
        </StackPanel>
      </DataTemplate>
   </ListBox.ItemTemplate>

</ListBox>
```

Source Code Snippet 9c

- The image is provided via a **HttpHandler** class implemented in the **ComparisSilverlightChallenge_Web** project, called **Image.ashx**. It reads the query string: **?ID=** from the HttpContext and returns the image associated with the specified car ID.
- To get from the car ID to the complete URL including the query string, we need a helper class. Add the **UrlBuilderConverter** class to **Page.xaml.cs**.
- Make sure the class is inside the **ComparisSilverlightChallenge** namespace.

```
public class UrlBuilderConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter,
      System.Globalization.CultureInfo culture)
   {
      return "http://localhost:53730/Image.ashx?ID=" + value.ToString();
   }

   public object ConvertBack(object value, Type targetType, object parameter,
      System.Globalization.CultureInfo culture)
   {
      return null;
   }
}
```

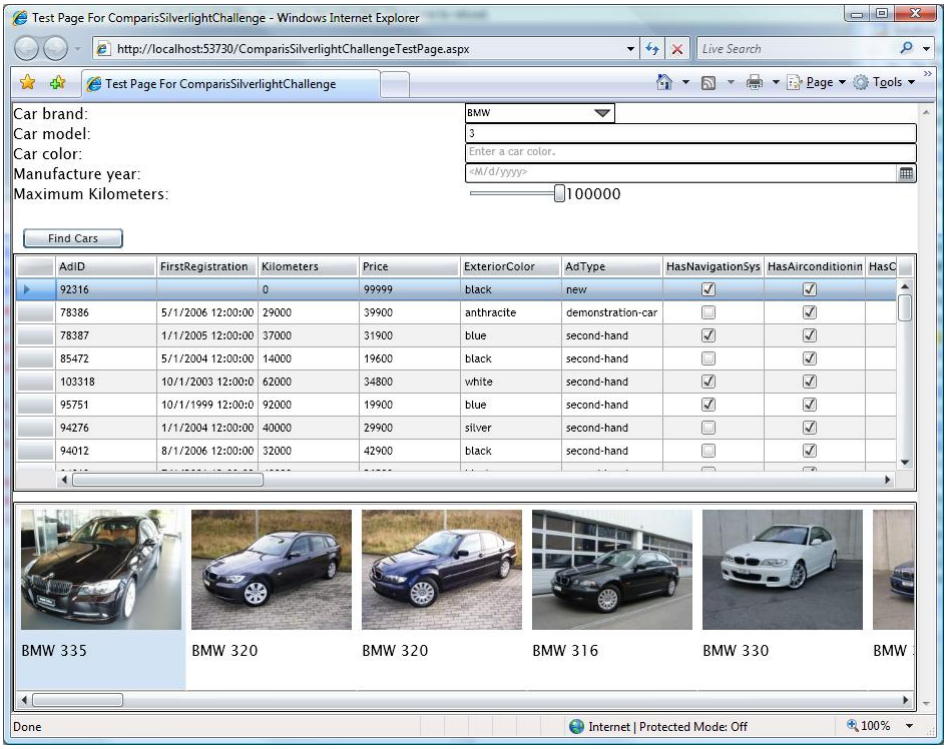
Source Code Snippet 9d

- **IValueConverter** needs a reference to **System.Windows.Data** to be added.
- In **Page.xaml**, before **LayoutRoot**, add the following resource declaration:

```
...
<UserControl.Resources>
   <ComparisSilverlightChallenge:UrlBuilderConverter x:Key="urlBuilderConverter"/>
</UserControl.Resources>

<Grid x:Name="LayoutRoot" Background="White">
...
```

Source Code Snippet 9e

- Save your work and run the project to see the new **ListBoxImages** control in action.



- Close the web browser.

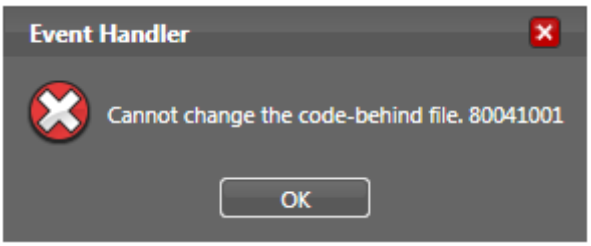
Step 10: Getting the SliderKilometers Control to work

Goal: When the user moves the slider, the Kilometer display **TextKilometers** right to the slider should be updated. When the user releases the slider, the currently displayed cars should be narrowed down to only those cars with the maximum amount of kilometers driven that is currently selected by the slider. For this, we need to handle two events, **ValueChanged** and **MouseLeftButtonUp**.

- In **Expression Blend**, select the **SliderKilometers** control and click on the **Events** button in the Properties window. Add these two event handlers:

Control	Event	Handler
SliderKilometers	MouseLeftButtonUp	SliderKilometers_Click
SliderKilometers	ValueChanged	SliderKilometers _ValueChanged

Note: If Expression Blend tells you that it is unable to edit the code behind file, you may still be running the application in Visual Studio's debugger which locks the source files. Stop the debugger by closing the web browser and try again.



- Back in **Visual Studio**, find the two event handlers generated by Expression Blend and add their logic:

```
 // Event handler: SliderKilometers value changed
 private void SliderKilometers_ValueChanged(object sender,
   RoutedPropertyChangedEventArgs<double> e)
 {
   if (TextKilometers != null)
   {
      TextKilometers.Text = ((int)e.NewValue).ToString();
   }
}
```

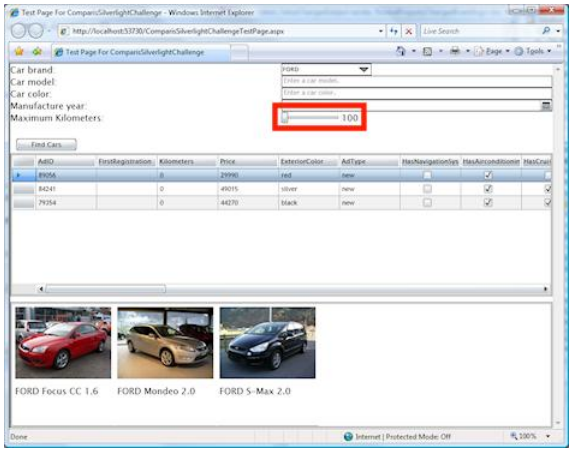
Source Code Snippet 10a

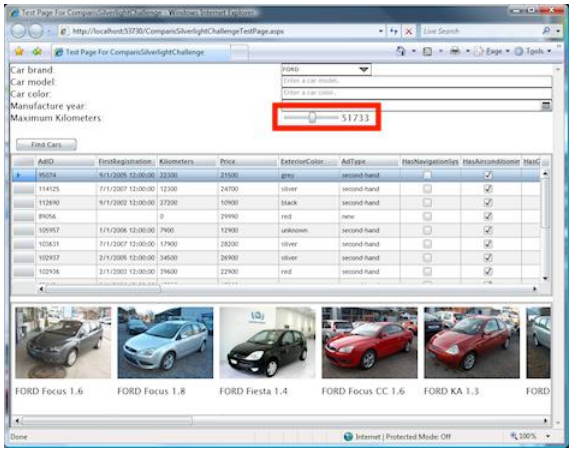
```
// Event handler: SliderKilometers Mouse button released
private void SliderKilometers_Click(object sender, MouseButtonEventArgs e)
{
   if (myAds != null)
   {
      var query = from ExAd in myAds
         where ExAd.Kilometers < SliderKilometers.Value
         select ExAd;

      if (query.Count() > 0)
      {
         DataGridCars.ItemsSource = query.ToList();
         ListBoxImages.ItemsSource = query.ToList();
      }
      else
      {
         DataGridCars.ItemsSource = null;
         ListBoxImages.ItemsSource = null;
      }
   }
}
```

Source Code Snippet 10b

Note: We changed the `RoutedEventArgs` to `RoutedPropertyChangedEventArgs<double>` in the **SliderKilometers_ValueChanged** event handler.

- Save your work and start the application. Search for cars and then use the slider control to narrow down the search in real-time.





- Close the web browser.

Step 11: Synchronizing DataGridCars and ListBoxImages

Goal: The two controls **DataGridCars** and **ListBoxImages** should always be synchronized so that when the selection in one is changed, the other control's selection should change as well.

- To do this, we need to capture the **SelectionChanged** event of both controls and add code that updates the control that was not changed.
- In **Expression Blend**, select both controls and use the **Event section** of the **Properties window** to add the events:

Control	Event	Handler
DataGridCars	SelectionChanged	DataGridCars _ SelectionChanged
ListBoxImages	SelectionChanged	ListBoxImages _ SelectionChanged

- Back in **Visual Studio** add the code to the event handlers that synchronizes the **SelectedItem property** of the two controls.
- For the ListBox control, change the `SelectionChangedEventArgs` to `EventArgs`.

```
// Event handler: DataGridCars selection is changed
private void DataGridCars_SelectionChanged(object sender, EventArgs e)
{
   ListBoxImages.SelectedItem = DataGridCars.SelectedItem;
}
```

Source Code Snippet 11a

```
// Event handler: ListBoxImages selection is changed
private void ListBoxImages_SelectionChanged(object sender, EventArgs e)
{
   DataGridCars.SelectedItem = ListBoxImages.SelectedItem;
}
```

Source Code Snippet 11b

Step 12: Adding a Popup Window with Car Details

Goal: Next we want to present the user with a popup window containing all the car's details and a larger image.

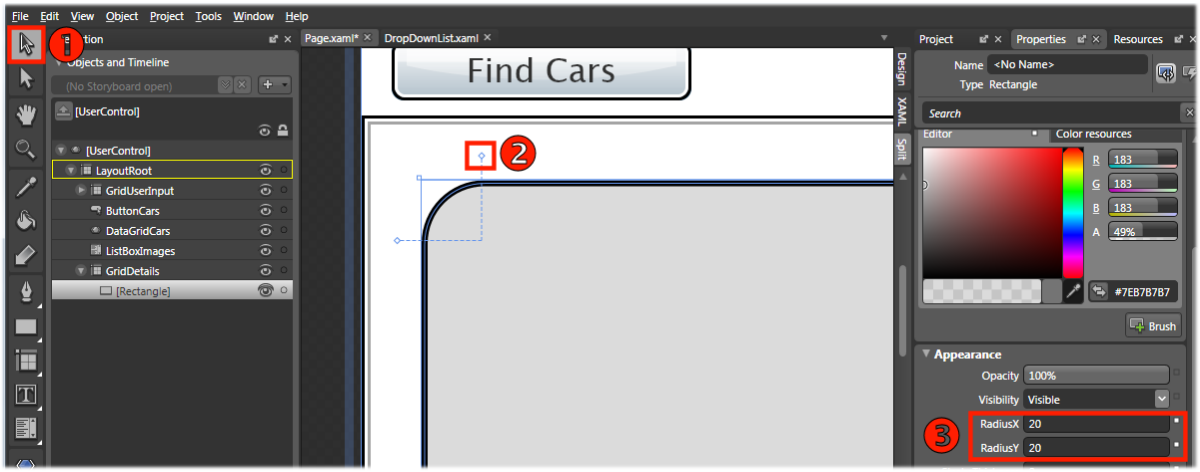
- Switch to **Expression Blend**, make **LayoutRoot** active and add a new **Grid** control:

Property	Value
Name	GridDetails
Width	Auto
Height	Auto
HorizontalAlignment	Stretch
VerticalAlignment	Stretch
Margin	20, 170, 20, 20

- Make **GridDetails** active and add a **Rectangle**. We don't have to give it a name as we won't be referencing it from code.

Property	Value
Width	Auto
Height	Auto
HorizontalAlignment	Stretch
VerticalAlignment	Stretch
Margin	0,0,0,0
Fill	#7EB7B7B7
Stroke	#FF000000
StrokeThickness	2
RadiusX	20
RadiusY	20

Hint: The **Radius** values can be set by using the selection tool ❶, holding the corners of the rectangle control and dragging it ❷. The Properties window displays the changes accordingly ❸.



- Add a **ListBox** control to **GridDetails**.

Property	Value
Name	ListBoxDetails
Width	350
Margin	15, 15, 0, 15
HorizontalAlignment	Left

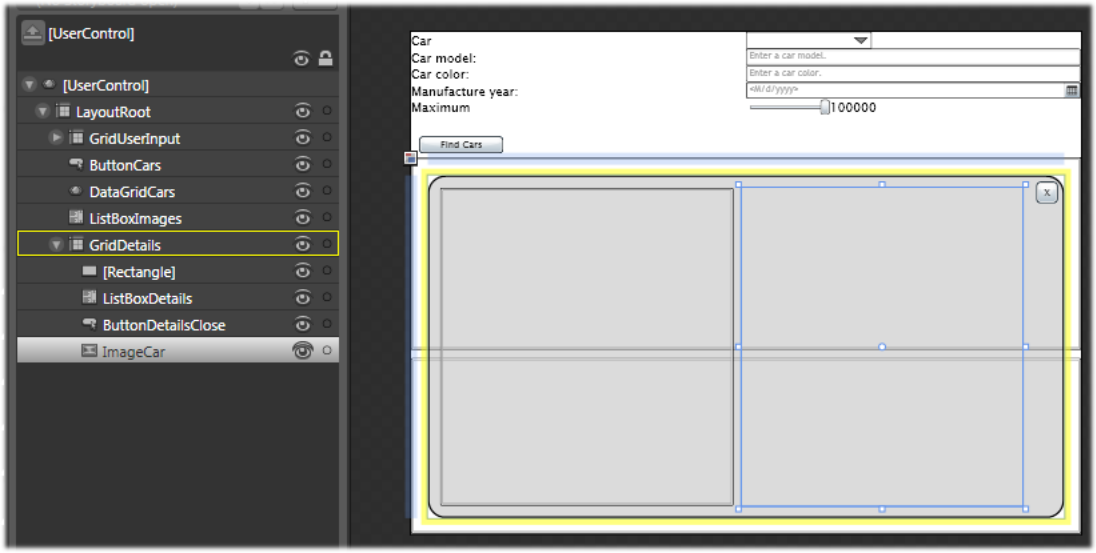
- Add a **Button** control to **GridDetails**.

Property	Value
Name	ButtonDetailsClose
Width	25
Height	25
HorizontalAlignment	Right
VerticalAlignment	Top
Margin	0,8,8,0
Content	X

- And add an **Image** control to **GridDetails**.

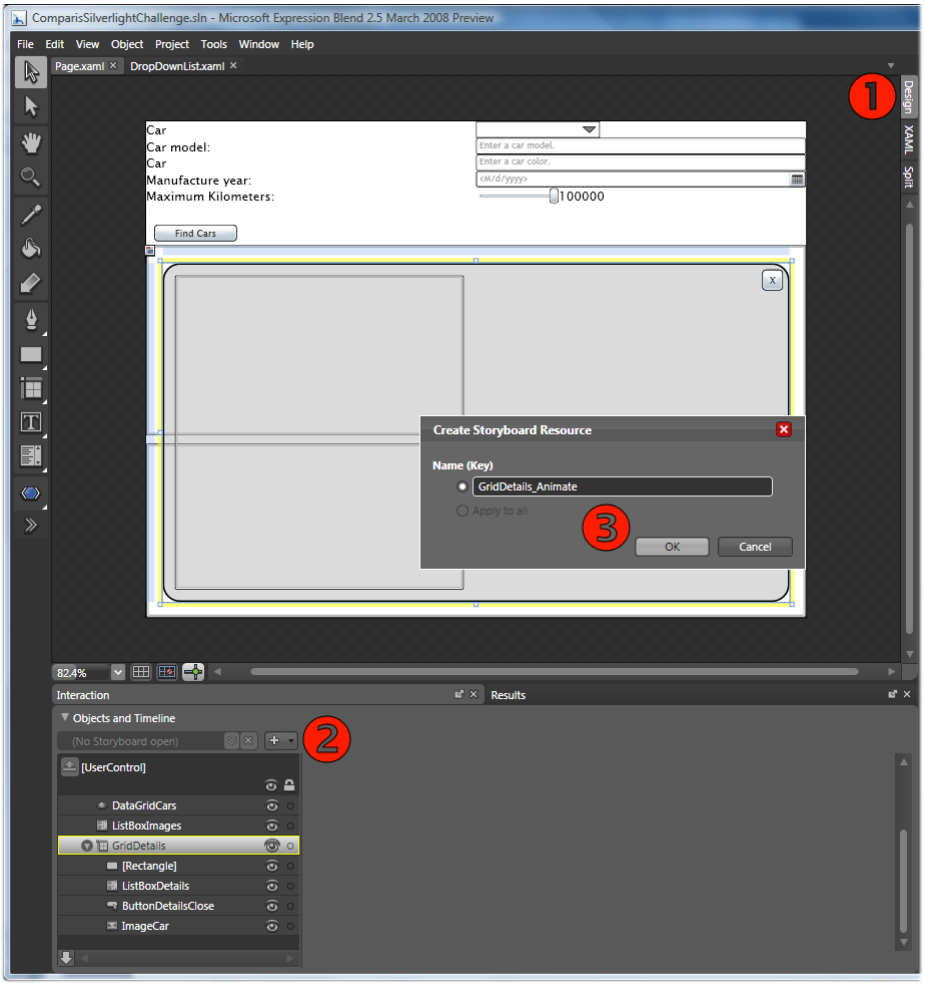
Property	Value
Name	ImageCar
HorizontalAlignment	Stretch
VerticalAlignment	Stretch
Margin	375,15,50,15

- Your Silverlight application should now look as follows

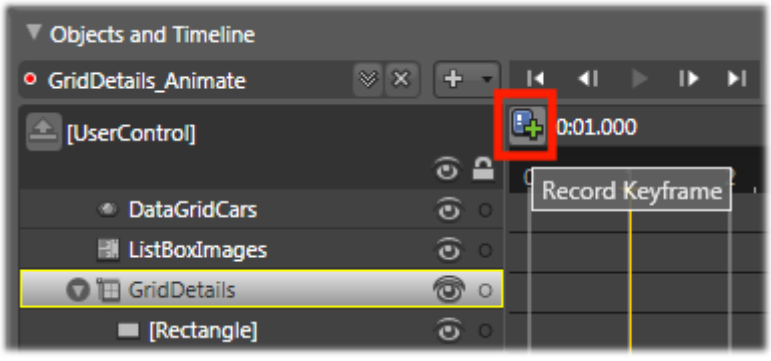


- We want **GridDetails** to be hidden when the application starts and we will add an animation that zooms the grid when it is displayed.

- Switch to the **Animation Workspace** using **Window**, **Active Workspace** menu or by hitting F6. Now hide the XAML code by clicking on the **Design** tab right of the designer area ❶.
- In the **Objects and Timeline** window select the **GridDetails** control and click the  icon ❷. In the dialog, name the new storyboard resource: **GridDetails_Animate** ❸.

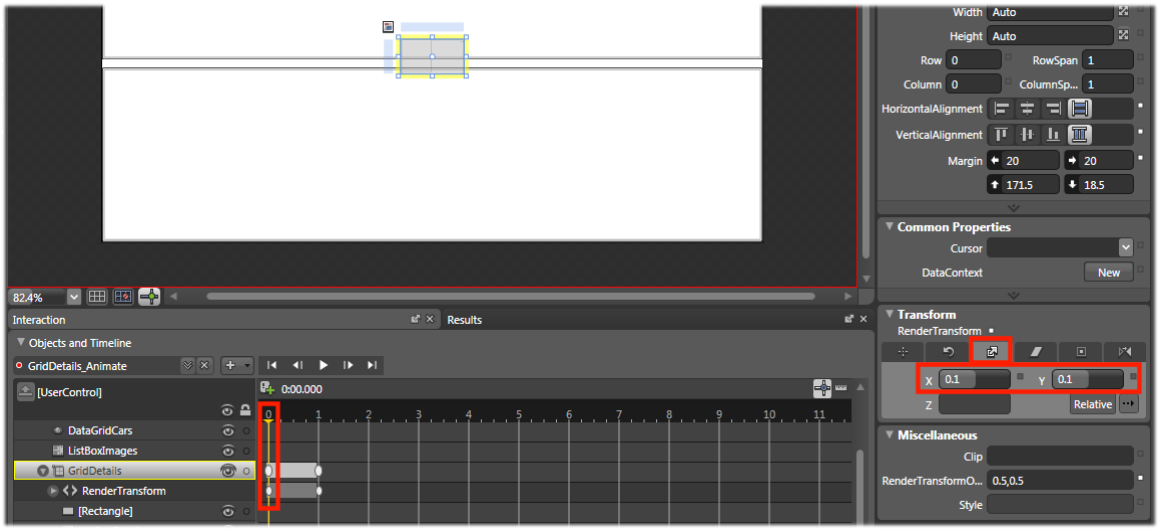


- With **GridDetails** still selected, advance the time slider in the **Objects and Timeline** window to the **1 second** marker and click the "**Record Keyframe**" button. This will record the current state of GridDetails at second 1.

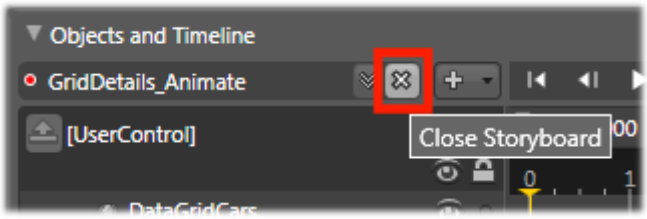


- Move the time slider to the **0 second** marker and in the **Properties window** under **Transform** select **Scale** and set

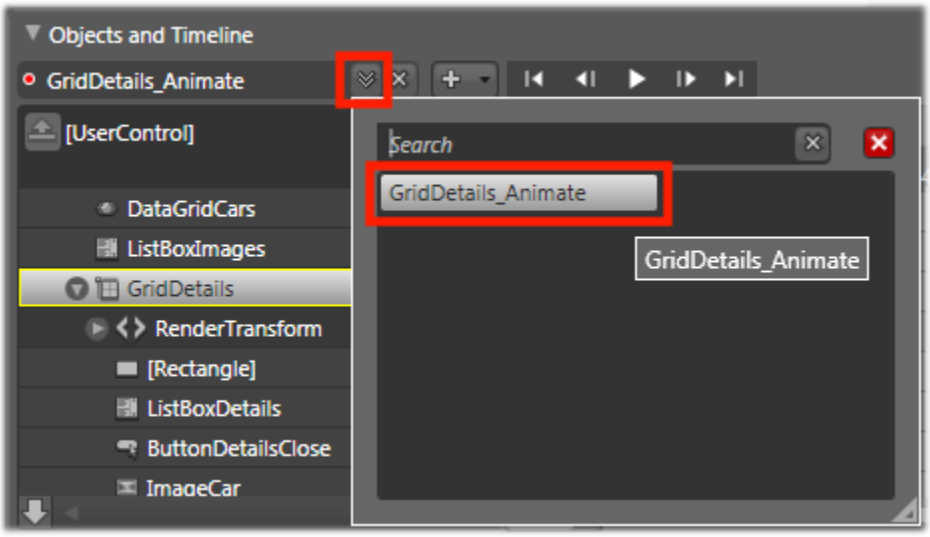
Property	Value
X	0.1
Y	0.1



- Click the **Close Storyboard** button and switch back to the **Design Workspace** (Hit F6)



Hint: You can always go back to editing and reviewing your storyboards by clicking the **storyboard-selector** in the **Animation Workspace** and choosing which storyboard you want to edit.



- We want to hide **GridDetails** by default, so set the **Visibility** property to "**Collapsed**".
- Add a **Button** that the user can click to show the new **GridDetails** control.

Hint: A fast way to do this is to copy the XAML code of the existing button and change the new Button control.

- In **Visual Studio**, edit the XAML source of the **Page.xaml** file and duplicate the first Button (x:Name="ButtonCars"). For the new Button, change the properties as follows:

Property	Value
Name	ButtonDetails
Width	100
Height	20
HorizontalAlignment	Left
VerticalAlignment	Top
Margin	125,125,0,0
Content	Get Car Details

- Add the event handler for the **ButtonDetails** control's **Clicked** event.

Control	Event	Handler
ButtonDetails	Click	ButtonDetails _ Click

- In the **ListBoxDetails** control, we want to show name- and value-pairs for each property in the **ExpandedAd** object. So we will pass it a list of objects containing each one item name and value. We will call this object "**Helper**", so let's implement its class:

```
public class Helper
{
   public string PropertyName { get; set; }
   public object PropertyValue { get; set; }
}
```

Source Code Snippet 12a

- Add the code to open the **GridDetails** control in the **ButtonDetails_Click** event handler:

```
// Event handler: ButtonDetails is clicked
private void ButtonDetails_Click(object sender, RoutedEventArgs e)
{
   ExpandedAd exAd = (ExpandedAd)DataGridCars.SelectedItem;

   if (exAd != null)
   {
      List<Helper> list = (from p in exAd.GetType().GetProperties() select new Helper
      {
         PropertyName = p.Name, PropertyValue = p.GetValue(exAd, null)
      }
      ).ToList();

      ListBoxDetails.ItemsSource = list;
      BitmapImage bmp = new BitmapImage(
         new Uri("http://localhost:53730/Image.ashx?ID=" + exAd.PictureID));
      ImageCar.Source = bmp;
      GridDetails_Animate.Begin();
      GridDetails.Visibility = Visibility.Visible;
   }
}
```

Source Code Snippet 12b

- **BitmapImage** needs a reference to **System.Windows.Media.Imaging**. Allow Visual Studio to add the using statement
- In the **ButtonCars_Click** event handler, we want to reset the **ItemsSource** of our **ListBoxDetails** in case we still have data from a previous search.

```
// Event handler: ButtonCars_Click
void ButtonCars_Click(object sender, RoutedEventArgs e)
{
   ListBoxDetails.ItemsSource = null;
...
```

Source Code Snippet 12c

- Add an event handler for the **Click** event of the **ButtonDetailsClose** control.

Control	Event	Handler
ButtonDetailsClose	Click	ButtonDetailsClose _ Click

- With the following logic:

```
// Event handler: ButtonDetailsClose is clicked
private void ButtonDetailsClose_Click(object sender, RoutedEventArgs e)
{
   GridDetails.Visibility = Visibility.Collapsed;
}
```

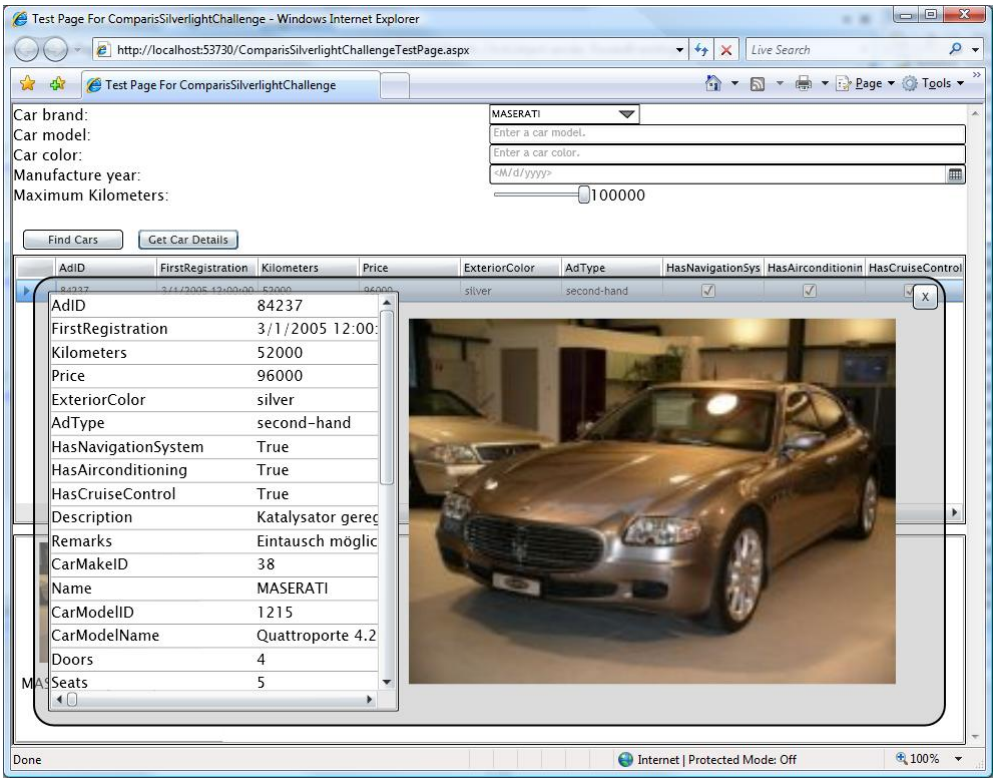
Source Code Snippet 12d

- The last thing we need to do to make our **GridDetails** work is to tell **ListBoxDetails** what data to show. Change **ListBoxDetails** in the XAML source by adding the following **ItemTemplate**:

```
<ListBox Margin="15,15,0,15" x:Name="ListBoxDetails" Width="350"
         HorizontalAlignment="Left">

   <ListBox.ItemTemplate>
      <DataTemplate>
         <StackPanel Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="{Binding PropertyName}"/>
            <TextBlock Margin="5,0,0,0" Text="{Binding PropertyValue}"/>
         </StackPanel>
      </DataTemplate>
   </ListBox.ItemTemplate>

</ListBox>
```

Source Code Snippet 12e

- Run the application and test the **ButtonDetails** and the **GridDetails** controls. When a car is selected, the button should open the details control, the close-button should hide it.



- Close the browser window.

Step 13: Working with Bookmarks

Goal: A very simple to implement but powerful feature in Silverlight is called **Bookmarks**. A bookmark gives you read- and write-access to URL parameters. If we update the URL with a parameter specifying the CarID every time the user selects a car in the car data grid or the car images list and look for the presence of this

parameter when the application loads, displaying the respecting car if it is present, the user can bookmark the browser's URL if he sees a car that he likes to share with friends or to return to at a later time.

- To write the bookmark, we need only add the following method:

```
private void PutBookmark()
{
   ExpandedAd exAd = (ExpandedAd)DataGridCars.SelectedItem;
   if (exAd != null)
      HtmlPage.Window.CurrentBookmark = exAd.AdID.ToString();
}
```

Source Code Snippet 13a

- ...and call it from the two event handlers that are fired when a selection changes, **DataGridCars_SelectionChanged** and **ListBoxImages_SelectionChanged**.

```
// Event handler: DataGridCars selection is changed
private void DataGridCars_SelectionChanged(object sender, EventArgs e)
{
   ListBoxImages.SelectedItem = DataGridCars.SelectedItem;
   this.PutBookmark();
}
```

Source Code Snippet 13b

```
// Event handler: ListBoxImages selection is changed
private void ListBoxImages_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
   DataGridCars.SelectedItem = ListBoxImages.SelectedItem;
   this.PutBookmark();
}
```

Source Code Snippet 13c

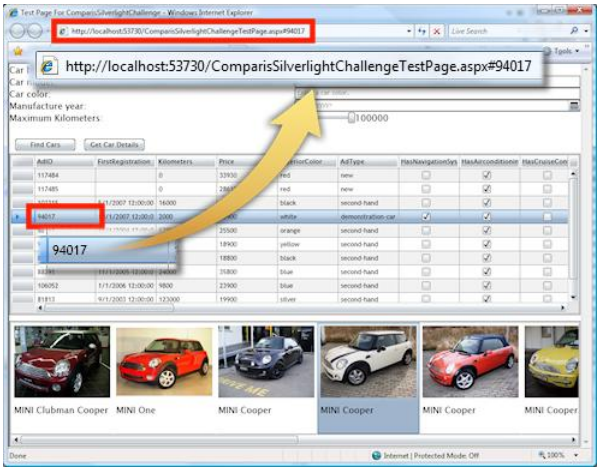
- To look for the bookmark and load it when the application starts, we add the following code to the **Page** constructor.

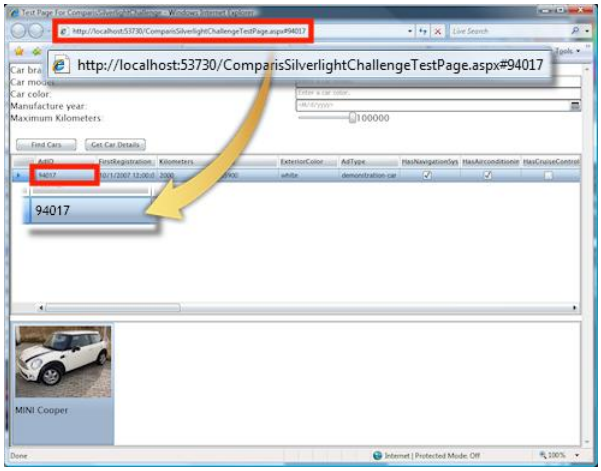
```
public Page()
{
   InitializeComponent();

   // Declare ButtonCars Click event handler
   ButtonCars.Click += new RoutedEventHandler(ButtonCars_Click);

   // Instantiate Web service
   CarfinderServiceClient uniqueCarMakes = new CarfinderServiceClient();

   // Populate DropDownBox control
   uniqueCarMakes.GetCarMakesCompleted += new
      EventHandler<GetCarMakesCompletedEventArgs>(uniqueCarMakes_GetCarMakesCompleted);
      uniqueCarMakes.GetCarMakesAsync("select * from CarMake order by Name ASC");

   // Read Bookmark and if found, query for the car in question
   if (HtmlPage.Window.CurrentBookmark != "")
   {
      StringBuilder sqlQuery = new StringBuilder();
      sqlQuery.AppendFormat("select * from ExpandedAd where AdID = '{0}'",
         int.Parse(HtmlPage.Window.CurrentBookmark));
```

```
        query_Cars(sqlQuery.ToString());
    }
}
```

Source Code Snippet 13d

- Run the project again and test the bookmarks. Search for a car, click on it in the data grid, copy the URL and paste it into a second browser window. The application should now load and display exactly the car that we selected before.





- Close the browser window.

Step 14: Adding a Template to the Data Grid

Goal: Previously, we set the data grid **DataGridCars** to automatically generate columns. If we want to manually select which columns are displayed, we can template it in a similar way that we have done with the **ListBoxes** before.

- Change the following properties of **DataGridCars**:

Property	Value
AutoGenerateColumns	False
RowHeight	40

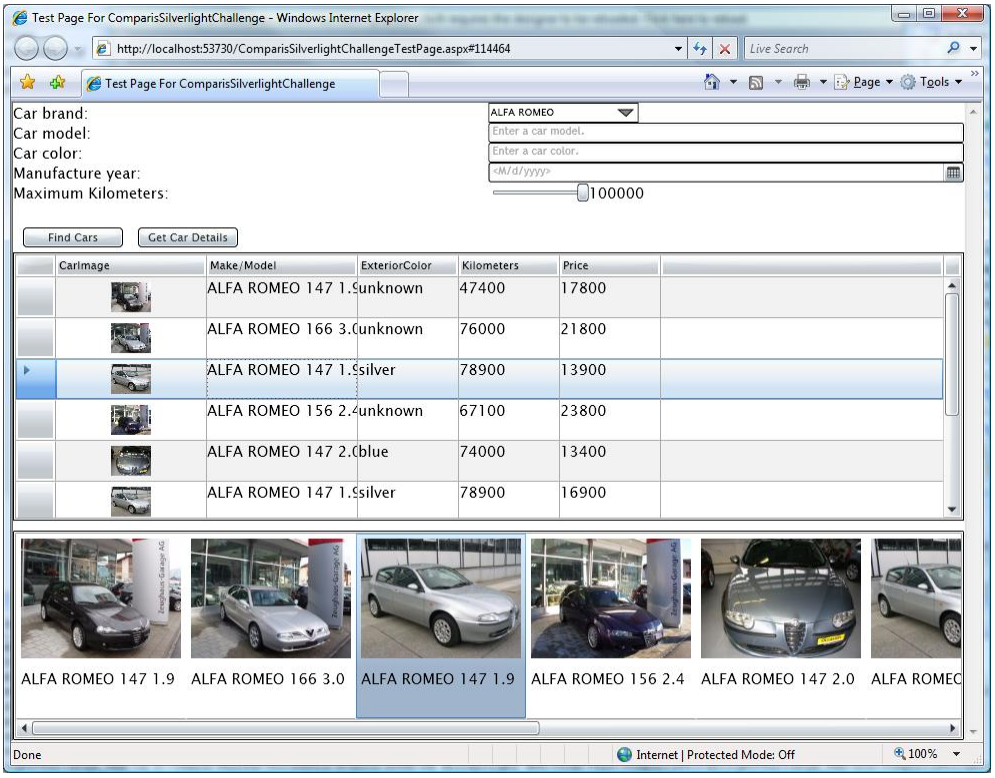
- and add (paste) the following code to the **DataGridCars** element in the XAML source of **Page.xaml**:

```
<System_Windows_Controls:DataGrid Margin="0,150,0,220" AutoGenerateColumns="False"
   x:Name="DataGridCars" SelectionChanged="DataGridCars_SelectionChanged" RowHeight="40" >

   <System_Windows_Controls:DataGrid.Resources>
      <Style TargetType="TextBlock">
         <Setter Property="FontSize" Value="5"></Setter>
      </Style>
   </System_Windows_Controls:DataGrid.Resources>

   <System_Windows_Controls:DataGrid.Columns>
```

```
      <System_Windows_Controls:DataGridTemplateColumn Width="150" Header="CarImage">
         <System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
               <Image Source="{Binding PictureID,
                     Converter={StaticResource urlBuilderConverter}}"
                     Margin="5" x:Name="AdImage" />
            </DataTemplate>
         </System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
      </System_Windows_Controls:DataGridTemplateColumn>

      <System_Windows_Controls:DataGridTemplateColumn Width="150" Header="Make/Model">
         <System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
               <StackPanel Orientation="Horizontal">
                  <TextBlock Text="{Binding Name}"/>
                  <TextBlock Text="{Binding CarModelName}" Margin="5,0,0,0"/>
               </StackPanel>
            </DataTemplate>
         </System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
      </System_Windows_Controls:DataGridTemplateColumn>

      <System_Windows_Controls:DataGridTemplateColumn Width="100" Header="ExteriorColor">
         <System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
               <TextBlock Text="{Binding ExteriorColor}" />
            </DataTemplate>
         </System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
      </System_Windows_Controls:DataGridTemplateColumn>

      <System_Windows_Controls:DataGridTemplateColumn Width="100" Header="Kilometers">
         <System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
               <TextBlock Text="{Binding Kilometers}" />
            </DataTemplate>
         </System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
      </System_Windows_Controls:DataGridTemplateColumn>

      <System_Windows_Controls:DataGridTemplateColumn Width="100" Header="Price">
         <System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
            <DataTemplate>
               <TextBlock Text="{Binding Price}" />
            </DataTemplate>
         </System_Windows_Controls:DataGridTemplateColumn.CellTemplate>
      </System_Windows_Controls:DataGridTemplateColumn>

   </System_Windows_Controls:DataGrid.Columns>

</System_Windows_Controls:DataGrid>
```

Source Code Snippet 14a

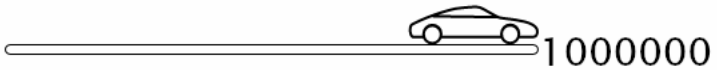
- Run the application to see the customized data grid in action, now also containing a small image of the car:



- Close the browser window.

Step 15: Styling a Control

Goal: The final thing that we want to do is to create a custom style for the slider control **SliderKilometers** in the shape of a car that you can move along a "road".

Before:	After:
	

- We are going to put the style into the **App.xaml** file so it becomes a global resource, available to all Silverlight controls in our application.

Note: Expression Blend today supports visual editing of control templates for WPF only, not for Silverlight. So we have to add the XAML code manually.

```
<Application xmlns="http://schemas.microsoft.com/client/2007"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   x:Class="ComparisSilverlightChallenge.App"
   >
   <Application.Resources>

      <Style x:Key="sliderStyle" TargetType="Slider">
         <Setter Property="Template">
            <Setter.Value>
```

```
            <ControlTemplate TargetType="Slider">
               <Grid x:Name="RootElement">
                  <Grid.Resources>
                     <ControlTemplate x:Key="RepeatButtonTemplate">
                        <Grid x:Name="RootElement"
                           Background="Transparent" Opacity="0" />
                     </ControlTemplate>
                  </Grid.Resources>
                  <Grid x:Name="HorizontalTemplateElement">
                     <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition Width="Auto" />
                     </Grid.ColumnDefinitions>
                     <Rectangle StrokeThickness=".5" Stroke="Black" Fill="#FFFFFFFF"
                        Grid.Column="0" Grid.ColumnSpan="3" Height="4"
                        RadiusX="2" RadiusY="2">
                     </Rectangle>
                     <RepeatButton
                        x:Name="HorizontalTrackLargeChangeDecreaseRepeatButtonElement"
                        Grid.Column="0" Template="{StaticResource RepeatButtonTemplate}"
                        IsTabStop="False" />
                     <Thumb x:Name="HorizontalThumbElement" Grid.Column="1"
                        Style="{StaticResource thumbStyle}"/>
                     <RepeatButton
                        x:Name="HorizontalTrackLargeChangeIncreaseRepeatButtonElement"
                        Grid.Column="2" Template="{StaticResource RepeatButtonTemplate}"
                        IsTabStop="False" />
                  </Grid>
               </Grid>
            </ControlTemplate>
         </Setter.Value>
      </Setter>
   </Style>
   <Style x:Key="thumbStyle" TargetType="Thumb">
      <Setter Property="Template">
         <Setter.Value>
            <ControlTemplate TargetType="Thumb">
               <Grid x:Name="RootElement">
                  <Grid.Resources>
                     <Storyboard x:Name="Normal State"/>
                     <Storyboard x:Name="Pressed State">
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Path"
                        Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
                           BeginTime="00:00:00" Duration="00:00:00.0010000">
                           <SplineColorKeyFrame KeyTime="00:00:00" Value="#FFFF0000"/>
                        </ColorAnimationUsingKeyFrames>
                     </Storyboard>
                  </Grid.Resources>
                  <Path x:Name="Path"  Margin="0,0,0,18" Width="48.6869" Height="14.4182"
                     Canvas.Left="149.028" Canvas.Top="385.791" Stretch="Fill"
                     StrokeStartLineCap="Round" StrokeEndLineCap="Round"
                     StrokeLineJoin="Round" Stroke="#FF000000" Fill="#FFFFFFFF"
                     Data="M 157.864,393.149C 159.676,393.149 161.144,394.617
161.144,396.429C 161.144,398.24 159.676,399.709 157.864,399.709C 156.053,399.709
154.584,398.24 154.584,396.429C 154.584,394.617 156.053,393.149 157.864,393.149 Z M
187.024,393.069C 188.836,393.069 190.304,394.537 190.304,396.349C 190.304,398.16
188.836,399.629 187.024,399.629C 185.213,399.629 183.744,398.16 183.744,396.349C
183.744,394.537 185.213,393.069 187.024,393.069 Z M 150.626,397.402C 149.346,396.842
149.082,393.361 150.386,392.682C 157.87,388.787 166.37,386.954 166.37,386.954C
166.37,386.954 173.697,385.894 175.537,386.454C 177.377,387.014 181.789,390.004
183.204,390.287C 185.287,390.704 192.407,391.107 195.287,392.787C 197.686,394.187
197.212,395.409 197.106,396.202C 196.946,397.402 195.346,397.402 195.346,397.402L
190.087,397.402C 190.254,397.008 190.346,396.576 190.346,396.122C 190.346,394.31
188.878,392.842 187.066,392.842C 185.255,392.842 183.786,394.31 183.786,396.122C
183.786,396.576 183.879,397.008 184.045,397.402L 160.96,397.402C 161.106,397.03
```

```
161.186,396.625 161.186,396.202C 161.186,394.39 159.718,392.922 157.906,392.922C
156.095,392.922 154.626,394.39 154.626,396.202C 154.626,396.625 154.707,397.03
154.853,397.402L 150.626,397.402 Z M 158.236,389.722L 182.236,390.055"/>
                </Grid>
              </ControlTemplate>
            </Setter.Value>
          </Setter>
      </Style>

   </Application.Resources>
</Application>
```

Source Code Snippet 15a

- Assign the slider style to **SliderKilometers** in **Page.xaml** and increase height and width of the control so our new style has enough room to display.

Property	Value
Style	{StaticResource sliderStyle}
Width	200

```
<Slider Style="{StaticResource sliderStyle}" Height="50" Width="200" LargeChange="10000"
   Maximum="100000" Minimum="100" SmallChange="1000" Value="100000"
   x:Name="SliderKilometers" MouseLeftButtonUp="SliderKilometers_Click"
   ValueChanged="SliderKilometers_ValueChanged"/>
```

Source Code Snippet 15b

- Increase the height of the surrounding stack panel **StackPanelSlider** as well.

Property	Value
Height	50

- Save your code and rebuild the application. This may be necessary in both Visual Studio and Expression Blend for both of them to accept the new style.

Note: In the current version, Visual Studio's XAML design view does not show pages with styled controls properly, so don't worry if your page looks wrong here. Expression Blend should pick up your style and show the control in its new look.

- Run the application and test the styled slider control in action.

The End.