



Symphony Framework Academy

Radio Button

Symphony Framework Academy

Introduction.

The Symphony Framework is a set of Synergy .NET assemblies that help you to build powerful Windows Presentation Foundation applications utilizing your Synergy Repository Structures and existing Synergy Language code.

This tutorial demonstrates how easy it is to implement Radio Button selection fields within your Windows Presentation Foundation user interface. The Radio Button control provides the user the ability to select a single entry from a pre-defined number of selections, all of which are visible. The available selections are built directly from the selection list contents stored against a field in your Synergy Repository. This tutorial will demonstrate how easy it is to utilise a Radio Button control in your WPF applications. Welcome to the Symphony Framework Academy!

To complete this tutorial your system must have the following components installed:

- Microsoft Visual Studio 2010 SP1 or higher
- The Symphony Framework is better experienced when using Synergy/DE version 10!
- Synergy DBL Integration for Visual Studio (same version as Synergy/DE).
- Microsoft .NET Framework 4.0 or higher.
- The Symphony Framework requires the Microsoft Expression Blend Software Development Kit (SDK) for .NET 4 which can be downloaded from <http://www.microsoft.com/en-us/download/details.aspx?id=10801>
- Symphony Framework 2.1.0.0 or higher.
 - Download from <http://symphonyframework.codeplex.com/releases>.
- CodeGen version 4.2.6 or higher
 - Download from <http://codegen.codeplex.com/releases>.

Tutorial Setup.

There are no initial setup requirements for this tutorial

Symphony Framework Academy

Empowering your Applications with Radio Button Selections.

This tutorial will walk you through the few steps required to define selection data and provide a Radio Button user selection capability.

The first task of this tutorial is to load Visual Studio. From the start menu locate and run Microsoft Visual Studio 2010.

- From the file menu select File→New→Project.
- Within the New Project dialog locate the Synergex\DE → Windows entry in the Installed Templates list (on the left of the dialog).
- In the project types list (center list in the dialog) locate the **WPF Application** entry.
- In the **Name** entry give the project a name of **RadioSelection**.
- So you can find the project when you continue through further tutorials it is recommended that you place the project in a folder in your “**My Documents**” area. Click the **Browse** button.
 - Navigate to your **My Documents → Visual Studio 2010 → Projects** folder.
 - If the **SymphonyAcademy** folder exists, select it.
 - If the **SymphonyAcademy** folder does not exist, click the **New folder** button to create a folder and name it **SymphonyAcademy**.
 - Click the **Select Folder** button.
- Uncheck the **Create directory for solution** check box.
- Click the **OK** button to create the project.

The project creation wizard will run and your WPF application will be created. Because our application will be utilizing the Symphony Framework our first task is to reference the Symphony Framework assemblies.

- From the **Project** menu, select the **Add Reference...** menu entry.
- From the **Add Reference** dialog, click on the **Browse** tab. Visual Studio may take a few moments to respond.
- Browse your local hard drive for the Symphony Framework assemblies. The default installation location is `C:\Program Files\Synergex\SymphonyFramework\Bin\`. If you have a 64bit operating system the default installation location is `C:\Program Files (x86)\Synergex\SymphonyFramework\Bin\`.
- From within the folder, highlight and select only the four required assemblies:
 - **SymphonyAdapter.dll**
 - **SymphonyConductor.dll**
 - **SymphonyCore.dll**
 - **SymphonyCrescendo.dll**

Symphony Framework Academy

- Click the **OK** button to add the assemblies to your project.

Now we need to reference the required .NET assemblies that let us handle user interface interactions.

- From the **Project** menu, select the **Add Reference...** menu entry.
- From the **Add Reference** dialog, click on the **.NET** tab. Visual Studio may take a few moments to respond.
- Search the list for the **System.Windows.Interactivity** assembly and ensure you select the framework 4.0 version.
- Click **OK** to add the assembly to the project.

Symphony Framework Academy

To build a Synergy based application we will fully utilize the Synergy Repository. The Synergy Repository contains the structure and file definitions that allow us to code generate much of the required code. We are going to create a Synergy Repository so we can define the required field types.

- Within Visual Studio, right click the **RadioSelection** project in the **Solution Explorer** window.
 - If you don't have the **Solution Explorer** window visible, from the **View** menu column, select the **Solution Explorer** entry.
- From the dropdown context menu select the **Add** entry and then select the **New Folder** entry.
- Enter a name of **Repository**.

Now we can define the required environment variables to define the location of our repository files.

- From the **Project** menu, select the **RadioSelection Properties...** entry.
- Click the **Environment Variables** tab.
- In the **Name** column, add a new entry called **RPSMFIL**.
- In the corresponding **Value** column, define the location of the repository main file. Enter the value `$(ProjectDir)Repository\rpsmain.ism`.
- In the **Name** column, add a new entry called **RPSTFIL**.
- In the corresponding **Value** column, define the location of the repository text file. Enter the value `$(ProjectDir)Repository\rpstext.ism`.
- Save the project files by selecting **Save All** on the **File** menu.
- It is recommended that you close and re-open Visual Studio at this time, and re-open the **RadioSelection** project to ensure that all the environment variables are correctly set.

Symphony Framework Academy

We can now create our new repository data files, define a structure and create the required fields that we are going to expose as Radio Button selection controls.

- Within Visual Studio, from the **Tools** menu column, select the **Synergy Repository** entry.
- Once in the Synergy Repository select the **Utilities** menu column, and then select the **Create New Repository**.
- Confirm the file specifications are correct. They should reference the following folder:
 - “My Documents\Visual Studio 2010\Projects\SymphonyAcademy\RadioSelection\Repository\”.
- Click the **OK** button.
- Click the **OK** button to confirm the creation of the repository data files.

Now we can create a new repository structure and define the required data fields.

- From the **Modify** column, select **Structures**.
- On the list, click the **Add** button.
- In the **Structure name** field enter a name of **RADIO**.
- Define the **File type** of **ASCII**.
- Enter a **Description** of **Radio selection example structure**.
- Select a **Tag type** of **None**.
- Click the **Attributes** button.

We are going to create a single field. The field will be a selection list and we will define the available selections within the repository.

- From the **Attributes** menu column select the **Fields** entry.
- From the list, click the **Add** button.
- Define the **Field name** as **OPTIONS**.
- Define the **Description** as **Selection list defined in the RPS**.
- From the available selections, define the **Type** as **Decimal**.
- Assign the **Size** field a value of **1**.
- Click on the **Display** tab.
- In the **Prompt** field enter the value **Door type**.
- Change the **View as** to **Radio buttons**.
- Click on the **Validation** tab.

Symphony Framework Academy

- Change the **Selections?** Entry to **List**.
- You will now be prompted for the available list entries.
 - Enter the first list item as **Standard**. Press the **CTRL+L** key combination to enter another list entry.
 - Enter the second list item as **Fire**. Press the **CTRL+L** key combination to enter another list entry.
 - Enter the third list item as **Grained**. Press the **CTRL+L** key combination to enter another list entry.
 - Enter the forth list item as **Smooth**. Press the **CTRL+L** key combination to enter another list entry.
 - Enter the fifth list item as **Glazed**. Click the **OK** button to complete the selections.
- Change the **Enumerated?** Selection to **Yes**.
- Enter a **Length** of **20**.
- Enter a **Base** of **1**.
- Enter a **Step** of **1**.
- Click the **OK** button.

This is our repository structure definition completed. You can close the Synergy Repository and confirm to save the changes!

Our first coding task is to create the Data Object class so we can define within the Symphony Framework the data elements.

- Within Visual Studio, right click the **RadioSelection** project in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** entry and then select the **New Folder** entry.
- Enter a name of **Model**.

In the **Model** folder we are going to create classes that define the required Data Objects.

We are going to code generate all the **Model** elements so we will create a script file to perform the code generation tasks.

- Within Visual Studio, right click the **Model** folder in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** menu entry, and then select the **New Item...** entry.
- In the **Installed Templates** view ensure that the **Synergy** entry is highlighted.
- In the **Items** list, locate the **Text File** entry.

Symphony Framework Academy

- Change the **Name** to **MakeModel.bat**.
- Click the **Add** button.

The code generation templates are stored in a folder under the Symphony Framework root folder. We are going to the “Symphony_Data” template which will code generate the required data object.

- In the **MakeModel.bat** script file define the logical **CODEGEN_TPLDIR** to reference the **%SYMPHONYTPL%** folder.
- Execute the **codegen** program passing the following command line options:
 - **-t Symphony_Data** *this defines the template to use.*
 - **-r** *denotes to replace any existing files.*
 - **-n RadioSelection.Model** *is the namespace declaration.*
 - **-prefix m** *defines the field prefix value*
 - **-s RADIO** *specifies the repository structure to use.*
- Save your changes.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_Data -r -n RadioSelection.Model -prefix m -s RADIO
```

From the **Tools** menu select the **Command Prompt** entry. This will open a command window.

- In the command window, navigate to the “**C:\Users\userName\Documents\Visual Studio 2010\Projects\SymphonyAcademy\RadioSelection\Model**” folder.
 - Replace *username* with your user name.
 - If you have not adopted the recommended folder structure, navigate to the **Model** folder under your main project folder.
- Execute the **MakeModel.bat** script.

The file will have been created by the code generator.

You can leave the command window open, but move back to Visual Studio.

- Within Visual Studio, right click the **Model** folder in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** menu entry, and then select the **Existing Item...** entry.
- Navigate to the **Model** folder.
- Click and highlight the generated file **Radio_Data.CodeGen.dbc**.
- Click the **Add** button.

Symphony Framework Academy

- To confirm all is in order you can perform a build of the application. You should not encounter any errors. If you do, resolve them. Close the **MakeModel.bat** tab window.

The Symphony Framework makes it very easy to build a rich user interface by using the Synergy Repository field definitions to style how input controls appear and how they function. To create these styles we will again use the repository definitions and code-generate the styles for each individual field.

- Within Visual Studio, right click the **RadioSelection** project in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** entry and then select the **New Folder** entry.
- Enter a name of **Resources**.

In the **Resources** folder we are going to create our “styles” resources. There are two classes we are going to code generate here. The first is the Content class. By utilizing the Synergy Repository we can identify fields that have selection lists associated with them and this code generated file will contain references to the selection items which will be presented as combo dropdown lists. This structure does not contain any fields that will be presented as a combo box but this file is required as a place holder. The second class is the Style resource. This file will define the UI appearance and data binding capabilities of each field in your repository structure.

We are going to code generate all the Resource elements so we will create a script file to perform the code generation tasks.

- Within Visual Studio, right click the **Resources** folder in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** menu entry, and then select the **New Item...** entry.
- In the **Installed Templates** view ensure that the **Synergy** entry is highlighted.
- In the **Items** list, locate the **Text File** entry.
- Change the **Name** to **MakeResources.bat**.
- Click the **Add** button.

The code generation templates are stored in a folder under the Symphony Framework root folder. We are going to use two templates. The first is called “Symphony_Content” which will identify all fields with selection lists assigned and generate the required resource references. The second is “Symphony_Style” which will code generate styles for each individual field for our chosen repository structure.

- In the **MakeResources.bat** script file define the logical **CODEGEN_TPLDIR** to reference the **%SYMPHONYTPL%** folder.

Symphony Framework Academy

- Execute the **codegen** program passing the following command line options:
 - **-t Symphony_Content** *this defines the template to use.*
 - **-r** *denotes to replace any existing files.*
 - **-n RadioSelection** *is the namespace declaration.*
 - **-ut ASSEMBLYNAME=RadioSelection** *specifies the assembly name*
 - **-s RADIO** *specifies the repository structure to use.*

- Execute the **codegen** program passing the following command line options:
 - **-t Symphony_Style** *this defines the template to use.*
 - **-r** *denotes to replace any existing files.*
 - **-n RadioSelection** *is the namespace declaration.*
 - **-ut ASSMELBYNAME=RadioSelection** *the project namespace*
 - **-s RADIO** *specifies the repository structure to use.*

- Save your changes.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%  
  
codegen -t Symphony_Content -r -n RadioSelection -ut ASSEMBLYNAME=RadioSelection -s RADIO  
codegen -t Symphony_Style -r -n RadioSelection -ut ASSEMBLYNAME=RadioSelection -s RADIO
```

Return to your command window.

- Navigate back up a folder and then into the Resources folder.
- Execute the **MakeResources.bat** script.

The files will have been created by the code generator.

You can leave the command window open, but move back to Visual Studio.

- Within Visual Studio, right click the **Resources** folder in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** menu entry, and then select the **Existing Item...** entry.
- Navigate to the **Resources** folder.
- Holding down the shift key, click and highlight the two generated files, **Radio_Content.CodeGen.xaml** and **Radio_Style.CodeGen.xaml**.
- Click the **Add** button.

There are additional steps required:

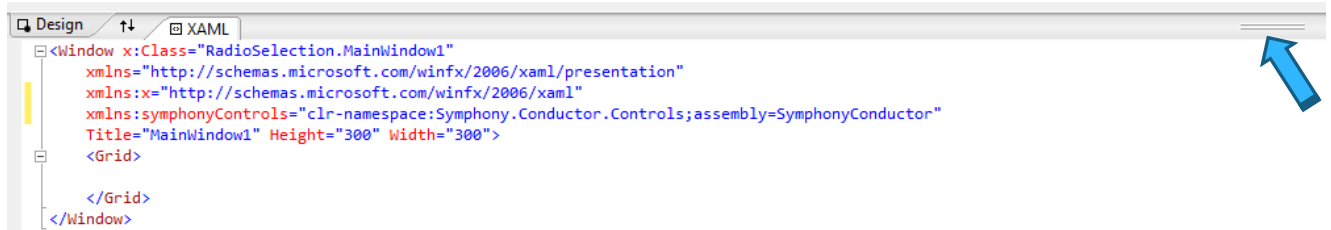
Symphony Framework Academy

- Highlight the recently added **Radio_Content.CodeGen.xaml** file. Right click and select **Properties**. Locate the **Build Action** in the list of properties and change this to be **Resource**.
- Highlight the recently added **Radio_Style.CodeGen.xaml** file. Right click and select **Properties**. Locate the **Build Action** in the list of properties and change this to be **Resource**.
- To confirm all is in order you can perform a build of the application. You should not encounter any errors. If you do, resolve them. Close the **MakeResources.bat** tab window.

We have now generated all the main elements that enable your Symphony Framework based application to present your repository based selections. We are now going to apply these to our user interface.

We will now write the XAML code that utilizes all the code generated elements and presents the available selection lists in the form of Combo Box control.

- Within Visual Studio, double click the **MainWindow1.xaml** file to bring the window into the visual designer.
- All our coding is going to be in the xaml editor. You can gain additional space by grabbing the splitter bar upwards to reveal a larger xaml editing region.



```
<Window x:Class="RadioSelection.MainWindow1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:symphonyControls="clr-namespace:Symphony.Conductor.Controls;assembly=SymphonyConductor"
        Title="MainWindow1" Height="300" Width="300">
    <Grid>
    </Grid>
</Window>
```

To utilize the Symphony Framework we first need to reference it within our xaml code.

- At the top of the xaml code, locate the **Title="MainWindow1"** attribute.
- Above this line, create a new blank line.
- Enter the namespace details that allows us to reference the Symphony Framework elements, **xmlns:symphonyControl="clr-namespace:Symphony.Conductor.Controls;assembly=SymphonyConductor"**.
 - o Please note this is all one line!
- locate the **Title="MainWindow1"** attribute and change its value to **"Radio Selection"**.
- Locate the **Width="300"** attribute and change its value to **"500"**.

Symphony Framework Academy

```
<Window x:Class="RadioSelection.MainWindow1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:symphonyControls="clr-namespace:Symphony.Conductor.Controls;assembly=SymphonyConductor"
  Title="Radio Selection" Height="300" Width="500">

  <Grid>

  </Grid>
</Window>
```

Now we need to add the resources we have code generated that allow us to locate and present the fields as a radio button control.

In the xaml editor, before the start <Grid> tag add a few blank lines to make the code more readable.

- Define the <Window.Resources></Window.Resources> tags.
- Within the <Window.Resources></Window.Resources> tags define the <ResourceDictionary></ResourceDictionary> tags.
- Within the <ResourceDictionary></ResourceDictionary> tags define the <ResourceDictionary.MergedDictionaries></ResourceDictionary.MergedDictionaries> tags.
- Within the <ResourceDictionary.MergedDictionaries></ResourceDictionary.MergedDictionaries> tags define the following references.
 - <ResourceDictionary Source="pack://application:,,,/SymphonyConductor;Component/Resources/Styles.xaml" />
 - <ResourceDictionary Source="pack://application:,,,/SymphonyConductor;Component/Resources/Converters.xaml" />
 - <ResourceDictionary Source="pack://application:,,,/RadioSelection;Component/Resources/Radio_Content.Codegen.xaml" />
 - <ResourceDictionary Source="pack://application:,,,/RadioSelection;Component/Resources/Radio_Style.Codegen.xaml" />

Your resource dictionary referencing code should look like this:

```
<Window.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="pack://application:,,,/SymphonyConductor;Component/Resources/Styles.xaml" />
      <ResourceDictionary Source="pack://application:,,,/SymphonyConductor;Component/Resources/Converters.xaml" />
      <ResourceDictionary Source="pack://application:,,,/RadioSelection;Component/Resources/Radio_Content.Codegen.xaml" />
      <ResourceDictionary Source="pack://application:,,,/RadioSelection;Component/Resources/Radio_Style.Codegen.xaml" />
    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
</Window.Resources>
```

Symphony Framework Academy

Now we can begin to utilize the code generated styles and content and build our user interface.

- Our next task is to create a grid layout. Please note that xaml is case sensitive. Allow the editor to assist you.
 - Within the `<Grid></Grid>` tags define the `<Grid.RowDefinitions></Grid.RowDefinitions>` tags.
 - Within the `<Grid.RowDefinitions></Grid.RowDefinitions>` tags add four `<RowDefinition>` tags
 - Define the height of the first row as `"*"`.
 - Define the height of the second row as `"auto"`.
 - Define the height of the third row as `"auto"`.
 - Define the height of the forth row as `"*"`.

Your xaml should now look like:

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

- Below the end `</Grid.RowDefintions>` tag, before the end `<Grid>` tag, we are going to define the column layout.
 - Define the `<Grid.ColumnDefinitions></Gird.ColumnDefinitions>` tags.
 - Within the `<Grid.ColumnDefinitions></Grid.ColumnDefinitions>` tags add five `<ColumnDefinition>` tags
 - Define the width of the first column as `"*"`.
 - Define the width of the second column as `"auto"`.
 - Define the width of the third column as `"auto"`.
 - Define the width of the forth column as `"auto"`.
 - Define the width of the fifth column as `"*"`.

Your xaml should now look like:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*"></ColumnDefinition>
  <ColumnDefinition Width="auto"></ColumnDefinition>
  <ColumnDefinition Width="auto"></ColumnDefinition>
  <ColumnDefinition Width="auto"></ColumnDefinition>
  <ColumnDefinition Width="*"></ColumnDefinition>
</Grid.ColumnDefinitions>
```

Symphony Framework Academy

We can now define the prompts and field controls that allow us to present and process the Combo selections.

- Below the end `</Grid.ColumnDefinition>` tag and above the end `</Grid>` tag add the following xaml.
- Define a `<Label>` control, setting the following attributes.
 - Set the **Grid.Row** to a value of `"1"`.
 - Define the **Grid.Column** to `"1"`.
 - Assign the **VerticalAlignment** property to a value of `"Center"`.
 - Assign the **Style** property a value of `"{StaticResource Radio_Options_prompt}"`.
 - Data bind the **DataContext** property to `"{Binding Path=PrimaryData}"`.
- Define a `<symphonyControls:FieldControl>` control, setting the following properties.
 - Set the **Grid.Row** to a value of `"1"`.
 - Define the **Grid.Column** to `"2"`.
 - Assign the **Style** property a value of `"{StaticResource Radio_Options_style}"`.
 - Data bind the **DataContext** property to `"{Binding Path=PrimaryData}"`.

The resulting xaml should look like:

```
<Label Grid.Row="1" Grid.Column="1" VerticalAlignment="Center"
      Style="{StaticResource Radio_Options_prompt}" DataContext="{Binding Path=PrimaryData}"></Label>
<symphonyControls:FieldControl Grid.Row="1" Grid.Column="2"
      Style="{StaticResource Radio_Options_style}"
      DataContext="{Binding Path=PrimaryData}">
</symphonyControls:FieldControl>
```

To ensure that our backing fields have the correct values in, we shall also display the “raw” storage data.

- Below previous lines of code add the following xaml.
- Define a `<TextBlock>` control, setting the following attributes.
 - Set the **Grid.Row** to a value of `"1"`.
 - Define the **Grid.Column** to `"3"`.
 - Specify a **Margin** of `"10,0,0,0"`.
 - Set the **VerticalAlignment** property to `"Center"`.
 - Assign the **Text** property to a value of `"{Binding Path=PrimaryData.Options}"`.

The resulting xaml should look like:

```
<!--now display the "raw" data-->
<TextBlock Grid.Row="1" Grid.Column="3" Margin="10,0,0,0"
          VerticalAlignment="Center"
          Text="{Binding Path=PrimaryData.Options}"></TextBlock>
```

Symphony Framework Academy

The final step is to create and data bind the View Model.

- Within Visual Studio, right click the **RadioSelection** project in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** entry and then select the **New Folder** entry.
- Enter a name of **ViewModel**.

In the **ViewModel** folder we are going to create a class that defines our view model.

- Within Visual Studio, right click the **ViewModel** folder in the **Solution Explorer** window.
- From the dropdown context menu select the **Add** menu entry, and then select the **New Item...** entry.
- In the **Installed Templates** view ensure that the **Synergy** entry is highlighted.
- In the **Items** list, locate the **Class** entry.
- Change the **Name** to **RadioViewModel.dbc**.
- Click the **Add** button.

The created class file will be added to the project and opened in the code editor.

- Above the *namespace* declaration we need to import the required Symphony Framework namespaces.
 - Import the **Symphony.Conductor.ViewModel** namespace.
 - Import the **RadioSelection.Model** namespace.

Your imports should look like:

```
import Symphony.Conductor.ViewModel
import RadioSelection.Model
```

- Move the cursor to the **public class** declaration. After the class name of **RadioViewModel** we need to extend the class by referencing the base **MaintenanceViewModel** class defined in the **Symphony.Conductor.ViewModel** namespace.
- Within the class definition we are going to create a constructor. The constructor is called when an instance of the class is created. To create the constructor we can use the **ctor** code snippet. Type **ctor <TAB><TAB>** to create the constructor code.
- The base **MaintenanceViewModel** class requires that we inject the “view” or “window” which the View Model object is being bound to. To enable this requirement we need to create a single parameter directly below the constructor method defining, above the *endparams* statement. The parameter has the following settings.
 - Make the parameter **inbound**.

Symphony Framework Academy

- Classify the parameter as **required**.
 - Name the parameter as **sender**.
 - Make the type **@System.Windows.FrameworkElement**.
- Below the *endparams* in the data division we can pass this parameter to the base `MaintenanceViewModel` base class constructor. We are also going to pass a new instance of our `Radio_Data` data object class. This is the data the window is data bound to.
- Call the base `MaintenanceViewModel` class constructor by using the keyword **parent**.
 - Pass the **sender** parameter as the first argument to the **parent** constructor.
 - Pass a **new** instance of the **Radio_Data()** class.

Your class code should look like:

```
public class RadioViewModel extends MaintenanceViewModel

    public method RadioViewModel
        in req sender ,@System.Windows.FrameworkElement
        endparams
        parent(sender, new Radio_Data())
    proc

    endmethod

endclass
```

The final step is to data bind our View Model to View – our `MainWindow1.xaml`.

- In the solution explorer expand the `MainWindow1` entry so that you can locate the `MainWindow1.xaml.dbl` file. Double click this file to bring it into the edit window.
- Above the namespace declaration add an **import** of the **RadioSelection.ViewModel**.
- Below the line that reads `this.InitializeComponent()` data bind the `ViewModel` by assigning the windows **DataContext** property to a **new** instance of the **RadioViewModel()** class. The **RadioViewModel** constructor requires the injection of the View, so pass in **this**.

The resulting code should look like:

```
import RadioSelection.ViewModel

namespace RadioSelection

    public partial class MainWindow1 extends Window

        public method MainWindow1
            endparams
        proc
            this.InitializeComponent()
            this.DataContext = new RadioViewModel(this)
        endmethod

    endclass

endnamespace
```


Symphony Framework Academy

You should now be able to build the completed solution. If you encounter any errors correct them. When you run the program the user interface will comprise of a radio button selection control. As you select items within the radio button control you will see the raw backing field data being updated.

Congratulations, you have completed the “Radio Button” tutorial. As you can see, with the Symphony Framework and the CodeGen tool you can rapidly create powerful user components that you can re-use throughout your application.