Symphony Framework Academy

Simple CRUD

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 build single file/single structure maintenance programs. The standard functions of Create/Read/Update/Delete (CRUD) are enabled and allow you to query, create and maintain data is a Synergy DBMS file. 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

Building your First C.R.U.D. Application.

This tutorial will walk you through the few steps required to build your first Symphony Framework empowered Windows Presentation Foundation (WPF) CRUD application. Once you have completed the tutorial you will have all the required elements needed to build a powerful application with a rich user interface.

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 Synergy\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 **PartMaint**.
- ☐ 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**

- ☐ 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.

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.

- ☐ From the **Project** menu, select the **PartMaint 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 Symphony Framework example repository main file, which can be referenced as $(SymphonyRPS)\rpsmain.ism
- ☐ In the **Name** column, add a new entry called **RPSTFIL**.
- ☐ In the corresponding **Value** column, define the location of the Symphony Framework example repository text file, which can be referenced as $(SymphonyRPS)\rpstext.ism

Save the project files by selecting **Save All** on the **File** column. To confirm your environment variables are set correctly, from the **Tools** menu column, select the **Synergy Repository** entry and confirm you repository files can be accessed by viewing the structure list. Close the Synergy Repository.

- ☐ It is recommended that you close and re-open Visual Studio at this time, and re-open the **PartMaint** project to ensure that all the environment variables are correctly set.

The Symphony Framework builds upon the Model-View-View Model (MVVM) design pattern. This technique of programming simply helps you to split your application into logical sections. The View section will contain the visual elements of your application. The Model section will contain your data and business logic and the View Model section will contain the elements that communication data and events between the View and the Model. This separation of concerns allows you to write powerful, scalable WPF applications. To help with the understanding of these separate concerns we will create folders within our project that will identify the three main MVVM areas.

- ☐ Within Visual Studio, right click the **PartMaint** 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 **Model**.

In the **Model** folder we are going to create our "model" entities. There are two classes we are going to code generate here. The first is the Data Object. By utilizing the Synergy Repository we can build structure aware Data Objects to manage data binding and expose our Synergy fields, through our View Model, to our View. The second class is a visibility class that allows us to control the fields visible to the user when searching for data items in the file.

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.
- ☐ 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 use two templates. The first is "Symphony_Data" which will code generate the Data Object for our chosen repository structure. The second is called "Symphony_Visibility" which defines a simple field visibility class that will allow users to hide/reveal fields for the file query lookup.

- ☐ 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 PartMaint.Model** *is the namespace declaration.*
- **–prefix m** *defines the repository include field prefix value.*
- **–s PART** *specifies the repository structure to use.*

☐ Execute the **codegen** program passing the following command line options:

- **–t Symphony_Visibility** *this defines the template to use.*
- **–r** *denotes to replace any existing files.*
- **–n PartMaint.Model** *is the namespace declaration.*
- **–s PART** *specifies the repository structure to use.*

☐ Save the file.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_Data -r -n PartMaint.Model -prefix m -s part
codegen -t Symphony_Visibility -r -n PartMaint.Model -s part
```

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\PartMaint\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 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 **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.

☐ Holding down the shift key, click and highlight the two generated files, **Part_Data.CodeGen.**dbc and **Part_Visibility.CodeGen.**dbc.

☐ Click the **Add** button.

☐ 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.

Our next task is to create the data IO classes that allow us to select and manage data in the Synergy DBMS data file.

- ☐ Within Visual Studio, right click the **PartMaint** 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 **DataIO**.

In the **DataIO** folder we are going to create our "data access" entities. There are two classes we are going to code generate here. The first is the FileIO class. By utilizing the Synergy Repository we can build structure aware file IO classes to access and manage data in our Synergy DBMS file. The second class is the Select class. This utilizes the Synergy Select class to facilitate the querying of data in the file.

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

- ☐ Within Visual Studio, right click the **DataIO** 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 **MakeDataIO.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 "Symphony_FileIO" which will code generate the file IO for our chosen repository structure. The second is called "Symphony_Select" which defines the query class.

- ☐ In the **MakeDataIO.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_FileIO** *this defines the template to use.*
  - **–r** *denotes to replace any existing files.*
  - **–n PartMaint.DataIO** *is the namespace declaration.*
  - **–s PART** *specifies the repository structure to use.*
- ☐ Execute the **codegen** program passing the following command line options:
  - **–t Symphony_Select** *this defines the template to use.*

- **–r** *denotes to replace any existing files.*
- **–n PartMaint.DataIO** *is the namespace declaration.*
- **–s PART** *specifies the repository structure to use.*

☐ Save the file.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_FileIO -r -n PartMaint.DataIO -s part
codegen -t Symphony_Select -r -n PartMaint.DataIO -s part
```

Return to your command window.

☐ Navigate back up a folder and then into the DataIO folder.

☐ Execute the **MakeDataIO.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 **DataIO** 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 **DataIO** folder.

☐ Holding down the shift key, click and highlight the two generated files, **Part_FileIO.CodeGen.**dbc and **Part_Select.CodeGen.**dbc.

☐ Click the **Add** button.

☐ 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 **MakeDataIO.bat** tab window.

The application we are writing needs to enable access to the data IO capabilities we have just created. Our next task is to create the application logic classes that allow us to perform the required data operations.

- ☐ Within Visual Studio, right click the **PartMaint** 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 **AppLogic**.

In the **AppLogic** folder we are going to create code that provides access from the program to the data IO facilities.

We are going to code generate the AppLogic element so we will create a script file to perform the code generation tasks.

- ☐ Within Visual Studio, right click the **AppLogic** 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 **MakeAppLogic.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 the "Symphony_CRUDAppLogic.tpl" template.

- ☐ In the **MakeAppLogic.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_CRUDAppLogic** *this defines the template to use.*
  - **–r** *denotes to replace any existing files.*
  - **–n PartMaint.AppLogic** *is the namespace declaration.*
  - **–s PART** *specifies the repository structure to use.*
- ☐ Save the file.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_CRUDAppLogic -r -n PartMaint.AppLogic -s part
```

Return to your command window.

- ☐ Navigate back up a folder and then into the **AppLogic** folder.
- ☐ Execute the **MakeAppLogic.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 **AppLogic** 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 **AppLogic** folder.
- ☐ Click and highlight the generated file, **Part_CRUDAppLogic.CodeGen.dbc**.
- ☐ Click the **Add** button.
- ☐ 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 **MakeAppLogic.bat** tab window.

The Symphony Framework utilizes many elements within the field definitions in your Synergy Repository. One element is the ability to expose fields assigned with selection list entries as Combo box controls. To enable this capability we need to expose collections that represent the data within these selection lists. Our next task is to create the collection classes that are built for all fields within the Synergy Repository that define selection list values. Even if your repository does not contain selection list fields you still need to complete this step so that the code generation process can build an empty place holder class.

- ☐ Within Visual Studio, right click the **PartMaint** 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 **Content**.

In the **Content** folder we are going to create code that provides collection classes for all fields defined with selection list entries.

We are going to code generate the Content element so we will create a script file to perform the code generation tasks.

- ☐ Within Visual Studio, right click the **Content** 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 **MakeContent.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 the "Symphony_Collection.tpl" template.

- ☐ In the **MakeContent.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_Collection** *this defines the template to use.*
  - **–r** *denotes to replace any existing files.*
  - **–n PartMaint.Content** *is the namespace declaration.*
  - **–s PART** *specifies the repository structure to use.*
- ☐ Save the file.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_Collection -r -n PartMaint.Content -s part
```

Return to your command window.

- ☐ Navigate back up a folder and then into the **Content** folder.
- ☐ Execute the **MakeContent.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 **Content** 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 **Content** folder.
- ☐ Click and highlight the generated file, **Part_Collection.CodeGen.dbc**.
- ☐ Click the **Add** button.
- ☐ 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 **MakeContent.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 **PartMaint** 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. 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.
- ☐ 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 PartMaint.Content** *is the namespace declaration.*
- **–ut ASSEMBLYNAME=PartMaint** *defines the project namespace.*
- **–s PART** *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 PartMaint** *is the namespace declaration.*
- **–ut ASSEMBLYNAME=PartMaint** *defines the project namespace.*
- **–s PART** *specifies the repository structure to use.*

☐ Save the file.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_Content -r -n PartMaint.Content -ut ASSEMBLYNAME=PartMaint -s part
codegen -t Symphony_Style -r -n PartMaint -ut ASSEMBLYNAME=PartMaint -s part
```

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, **Part_Content.CodeGen.xaml** and **Part_Style.CodeGen.xaml**.

☐ Click the **Add** button.

There are additional steps required:

☐ Highlight the recently added **Part_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 **Part_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.

To communicate data between the user interface and the program data and logic we utilize a class called a View Model. The View Model class exposes the data to the view and the events and data changes to the model. We are going to code generate the required View Model classes. Although we are only creating a single maintenance program we actually need two View Model class files. One view model class will be for the maintenance program and the other will be for the lookup element we are going to provide.

- ☐ Within Visual Studio, right click the **PartMaint** 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 our "View Model" classes. There are two classes we are going to code generate here. The first is the lookup class. The lookup class utilizes a base class in the Symphony Framework that performs the required query on the Synergy DBMS data file and loads the results table on a background worker thread. By doing this we do not block the UI, and also allow the user to cancel the query if they require. The second View Model class is for the actual maintenance program.

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

- ☐ 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 **Text File** entry.
- ☐ Change the **Name** to **MakeViewModel.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 "Symphony_BLListViewModel" which will code generate the back ground list view model for our chosen repository structure. The second is called "Symphony_CRUDViewModel" which Controls the maintenance functions of the program.

- ☐ In the **MakeViewModel.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_BLListViewModel** *this defines the template to use.*
  - **–r** *denotes to replace any existing files.*
  - **–n PartMaint.ViewModel** *is the namespace declaration.*
  - **–s PART** *specifies the repository structure to use.*
- ☐ Execute the **codegen** program passing the following command line options:
  - **–t Symphony_CRUDViewModel** *this defines the template to use.*
  - **–r** *denotes to replace any existing files.*
  - **–n PartMaint.ViewModel** *is the namespace declaration.*
  - **–s PART** *specifies the repository structure to use.*
- ☐ Save your changes.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_BLListViewModel -r -n PartMaint.ViewModel -s part
codegen -t Symphony_CRUDViewModel -r -n PartMaint.ViewModel -s part
```

Return to your command window.

- ☐ Navigate back up a folder and then into the **ViewModel** folder.
- ☐ Execute the **MakeViewModel.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 **ViewModel** 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 **ViewModel** folder.
- ☐ Holding down the shift key, click and highlight the two generated files, **Part_BLListViewModel.CodeGen.dbc** and **Part_CRUDViewModel.CodeGen.dbc**.

- ☐ Click the **Add** button.
- ☐ 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 **MakeViewModel.bat** tab window.

Defining the user interface is achieved by creating the various elements. Our single structure maintenance program comprises of a number of elements. We defined each area into its component parts. For example the maintenance aspect has a "key" view to allow the user to locate the required record. Then a "data" control to allow entry of the field data. There is a "query" control, which actually comprises of two controls – the criteria and the results. All of these elements make up the "view" elements. We are going to code generate all of these components.

- ☐ Within Visual Studio, right click the **PartMaint** 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 **View**.

In the **View** folder we are going to create our "View" classes. There are a number of classes we are going to build.

- ☐ Within Visual Studio, right click the **View** 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 **MakeView.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 a number of templates.

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

To create the key entry view:

- ☐ Execute the **codegen** program passing the following command line options:
    - **–t Symphony_KeyView** *this defines the template to use.*
    - **–r** *denotes to replace any existing files.*
    - **–n PartMaint.View** *is the namespace declaration.*
    - **–ut ASSEMBLYNAME=PartMaint** *defines the project namespace.*
    - **–s PART** *specifies the repository structure to use.*

To create the data entry view:

☐ Execute the **codegen** program passing the following command line options:
- **–t Symphony_DataView_WithGrid** *this defines the template to use.*
- **–r** *denotes to replace any existing files.*
- **–n PartMaint.View** *is the namespace declaration.*
- **–ut ASSEMBLYNAME=PartMaint** *defines the project namespace.*
- **–s PART** *specifies the repository structure to use.*

To create the query criteria entry view:

☐ Execute the **codegen** program passing the following command line options:
- **–t Symphony_CriteriaView_WithGrid** *this defines the template to use.*
- **–r** *denotes to replace any existing files.*
- **–n PartMaint.View** *is the namespace declaration.*
- **–ut ASSEMBLYNAME=PartMaint** *defines the project namespace.*
- **–s PART** *specifies the repository structure to use.*

To create the query list results view:

☐ Execute the **codegen** program passing the following command line options:
- **–t Symphony_ListView** *this defines the template to use.*
- **–r** *denotes to replace any existing files.*
- **–n PartMaint.View** *is the namespace declaration.*
- **–ut ASSEMBLYNAME =PartMaint** *defines the project namespace.*
- **–s PART** *specifies the repository structure to use.*

To create the query lookup view:

☐ Execute the **codegen** program passing the following command line options:
- **–t Symphony_LookUpView** *this defines the template to use.*
- **–r** *denotes to replace any existing files.*
- **–n PartMaint.View** *is the namespace declaration.*
- **–ut ASSEMBLYNAME =PartMaint** *defines the project namespace.*
- **–s PART** *specifies the repository structure to use.*

☐ Save the file.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_KeyView -r -n PartMaint.View -ut ASSEMBLYNAME=PartMaint -s part
codegen -t Symphony_DataView_WithGrid -r -n PartMaint.View -ut ASSEMBLYNAME=PartMaint  -s part
codegen -t Symphony_CriteriaView_WithGrid -r -n PartMaint.View -ut ASSEMBLYNAME=PartMaint  -s part
codegen -t Symphony_ListView -r -n PartMaint.View -ut ASSEMBLYNAME=PartMaint  -s part
codegen -t Symphony_LookUpView -r -n PartMaint.View -ut ASSEMBLYNAME=PartMaint  -s part
```

Return to your command window.

- ☐ Navigate back up a folder and then into the **View** folder.
- ☐ Execute the **MakeView.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 **View** 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 **View** folder.
- ☐ ***Ensure that the file filter choice is set to "All Files (*.*)"
- ☐ Holding down the shift key, click and highlight all the generated files.
- ☐ Click the **Add** button.
- ☐ 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 **MakeView.bat** tab window.

The final task is to code generate the contents of the MainWindow1.xaml (and .xaml.dbl) files that the project creation wizard created. This main window is the one executed and displayed first. We are going to use this to present and manage our code generated UI controls and to define the required visual states.

- ☐ Within Visual Studio, right click the **PartMaint** project 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 **MakeWindow.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 a number of templates.

- ☐ In the **MakeWindow.bat** script file define the logical **CODEGEN_TPLDIR** to reference the **%SYMPHONYROOT%\template** folder.
- ☐ Execute the **codegen** program passing the following command line options:
    - **–t Symphony_CRUDMainWindow** *this defines the template to use.*
    - **–r** *denotes to replace any existing files.*
    - **–n PartMaint** *is the namespace declaration.*
    - **–ut ASSEMBLYNAME=PartMaint**
    - **–ut WINDOWMINWIDTH=800**
    - **–ut WINDOWMINHEIGHT=450**
    - **–ut VIEWMODELNAMESPACE=PartMaint.ViewModel**
    - **–ut VIEWNAMESPACE=PartMaint.View**
    - **–s PART** *specifies the repository structure to use.*
- ☐ Save your changes.

Your script file should contain:

```
set CODEGEN_TPLDIR=%SYMPHONYTPL%

codegen -t Symphony_CRUDMainWindow -r -n PartMaint
    -ut ASSEMBLYNAME=PartMaint -ut WINDOWMINWIDTH=800
    -ut WINDOWMINHEIGHT=450 -ut VIEWMODELNAMESPACE=PartMaint.ViewModel
    -ut VIEWNAMESPACE=PartMaint.View -s part
```

*Note, your command should be all on one line

Return to your command window.

- ☐ Navigate back up a folder.
- ☐ Execute the **MakeWindow.bat** script.

The files will have been created by the code generator.

You can close the window. Return to Visual Studio.

Because the files we have just code generated have replaced existing files there is no requirement to add them to the project. Simply build you solution.

You should be able to execute the completed solution.

Look for the "💡" hints. Hovering over these will provide hints to the options available in that area of the program.