Creating Silverlight Applications with mv.NET

A product from BlueFinity

Copyright Notices

Document ref: mvNET_SL_DG
Revision 4.5.0

Contacting Us

We are always very happy to be able to discuss all aspects of our products with our customers - prospective and current alike. You can contact us via the following means:

Website:	www.bluefinity.com
Email:	support@bluefinity.com
Address:	10260 SW Greenburg Road, Suite 700, Portland, OR 97223, USA
Address:	Hamilton House, 111 Marlowes, Hemel Hempstead, Herts, HP1 1BB, UK

Trademark Acknowledgements

Contents

Welcome to mv.NET 1

The mv.NET Family of Products 1
Feature Overview 2
The mv.NET Suite 2
Developer Guide Contents 2
Assumptions 1

Silverlight Technical Overview 2

Silverlight Fundamentals 2
Developing Silverlight Applications 4

Development Prerequisites 5

Software Requirements 5
Solution Objects Entity Model 5

Creating Your First Silverlight Application 7

Create a New Solution 7
Add Assembly References 7
Web Site Project References 7
Browser Project References 8
Create a Web Service 8
Create a Web Service Reference 10
The XAML Wizard 10
Summary 10

Deploying Silverlight Applications 11

Silverlight Application Deployment Overview 11
mv.NET Specific Issues 11

The XAML Wizard 13

XAML Wizard Feature Summary 13

Invoking the XAML Wizard 14
Creating Form Designs 14
Form Designer Layout 15
Form Designer Regions 17
Re-arranging Region Content 19
Adding Controls to Regions 20
General Settings 21
Data Binding Settings 24
The Application Navigation Designer 26
Application Navigation Designer Layout 27
Treeview-based Application Navigation Design 28
Text Editor-based Application Navigation Design 28
Code Snippets 29
Code Generation 30
Creating a New Visual Studio Solution 31
General Notes on Using Controls 32
ComboBox 32
AutoCompleteBox 32
Image 33

Using a Business Access Layer within Silverlight 35

Why are Things Different in Silverlight 35
Solution Objects and Asynchronous Calling 36
An Example of Database Access 36
Calling Database Subroutines 38
The DataRepository Class in Silverlight 40
Using the XAML Wizard 40
Generating New Item IDs 40

Using the DataRepository Class 42

Constructing DataRepository Instances 42
The LoadData Method 43
DataRepository Events 44

Using the NavigationAssistant Class 45

Introduction to the NavigationAssistant Class 45
Managing the Display of Application Forms 46
OpenForm 46
ShowForm 46
CloseForm 47
ActiveForms 47

FormIsActive 47
Using Popup Windows 48

The Sample Silverlight Application 51

Sample Application Location 51
Pre-requisites for Running the Sample Application 51
Installing the Sample Application's EMR Definition 52
Sample Application Solution Structure 52
XAML Wizard Designs used in the Application 53

Welcome to mv.NET

Firstly, thank you for either purchasing one or more of the mv.NET products or for taking the time to explore the great functionality that they can provide to you and your fellow developers.

This chapter outlines the members of the mv.NET family of products and summarizes the contents of this guide.

The mv.NET Family of Products

mv.NET is ***the*** essential tool for any MultiValue database developer wishing to create .NET based application interfaces to their current or new MultiValue database file system.

The design goal of mv.NET is to enable the MultiValue developer to combine the power and flexibility of proven MultiValue technology with the state-of-the art, feature rich .NET environment. Its design also enables and encourages the developer to leverage, wherever possible, previously acquired MultiValue skills.

BlueFinity's team of software engineers has huge knowledge and experience of using both MultiValue systems and the .NET environment. We proudly regard ourselves as being one of the foremost companies in providing this technology bridge and look forward to working with you to enable you to meet your software development goals.

Feature Overview

The Silverlight integration components that are supplied as part of the standard mv.NET product include:

- Extensions to the Solution Objects entity modeling tool to allow the generated business access layer to be used inside the Silverlight environment.
- A XAML wizard to allow quick creation of applications menus and data maintenance forms.
- Silverlight-specific components to speed up your Silverlight application development activities

The mv.NET Suite

The mv.NET suite of products comprises:

- **Core Objects** - object oriented native .NET access to MultiValue databases.
- **Solution Objects** - Strongly-typed class-based access to your MultiValue database.
- **Adapter Objects** - complete implementation of an ADO.NET managed data provider for MultiValue databases, offering a standardized interface to database access.

Developer Guide Contents

The contents of this guide are designed to provide a basis for learning about how mv.NET can be used to create browser-based interfaces to your MultiValued database using Microsoft's Silverlight application framework.

Assumptions

This guide, through necessity, makes some assumptions about your skill level and software install base. Specifically, it assumes that:

1. You have already installed and configured mv.NET to connect into your database server. Details on how to do this can be found in the accompanying Getting Started and Core Objects guides.
2. You have created the necessary extended dictionary definitions for your data files. Details on how to do this can be found in the accompanying Core Objects developer guide.
3. You have created an entity model using the Solution Objects component of mv.NET. Details on how to do this can be found in the accompanying Solution Objects developer guide.
4. You have installed Microsoft Visual Studio 2010 and that you are reasonably familiar with its layout and general functioning.

Silverlight Technical Overview

Silverlight is a powerful application framework supplied by Microsoft. One of its key strengths is its ability to act as a superb a delivery platform for Rich Internet Applications (RIA) – in other words, it allows you to create rich client applications that run inside a web browser.

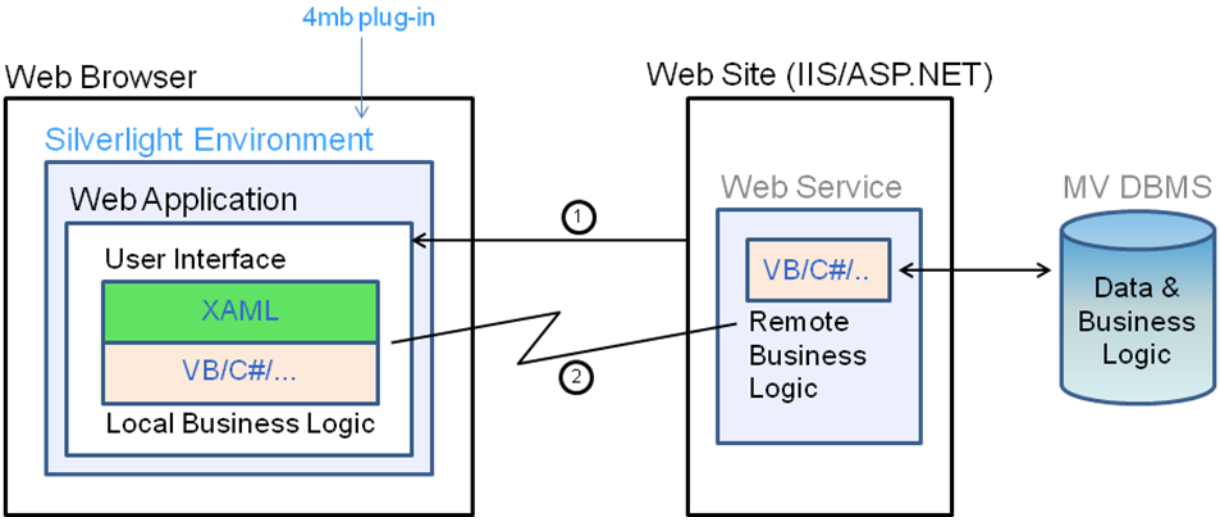
Although Silverlight is a Microsoft technology and has its own extensive documentation, this chapter provides a brief overview of the technology to set the backdrop for the rest of this guide.

Silverlight Fundamentals

Silverlight, described concisely, is a 5mb web browser plug-in that provides an execution environment for programs written using a subset of the .NET framework and WPF subsystem.

Thus, Silverlight allows developers to create applications using Visual Studio in a very similar manner to the way in which they create rich-client applications. It also means that the end-user experience of utilizing such an application is very similar to that of using a traditional desktop application.

Silverlight applications are typically split into 2 discrete sections. A section that runs inside the browser and a section that is hosted within an associated web service. The diagram below summarizes this segmentation:



1. The application is initially loaded via the web site
2. Access to database and other non-browser hosted resources is done via a web service

In the above diagram, the Silverlight application interface section (hosted within the web browser) comprises 2 parts – one created using XAML, the other using a .NET managed programming language, typically VB or C#.

XAML stands for eXtensible Application Markup Language. It is a way for developers to define the content and layout of application interfaces in a way which provides a clear separation of interface design from underlying business logic. As its name suggests, it is a markup language and as such it is XML-like in syntax. Visual Studio has a WYSIWYG visual designer which allows you to create XAML via an easy to use graphical designer – or, you can write it manually – or, typically, do a bit of both. mv.NET offers a further option for creating XAML content via its "XAML Wizard".

Thus, the Silverlight environment runs the interface as defined in XAML as well as the immediate business logic/interface event handling code locally within the browser. This promotes highly responsive, visually appealing and very user-friendly application interfaces.

Closely associated with a Silverlight application will be the parent web site. This will typically be the web site to which the user initially navigates to launch the application. This web site hosts a web service which can provide the Silverlight environment with access to resources located outside of the Browser environment, such as one or more database servers. Using mv.NET, this will be your backend MultiValued database.

Developing Silverlight Applications

If you are going to create Silverlight applications, you need to use Visual Studio 2010. It offers a far superior development experience when working with Silverlight when compared with Visual Studio 2008.

When creating Silverlight applications with Visual Studio, the 2 parts (as described above) are represented as 2 separate projects within a single solution. One project being a web site project the other being the part that runs within the Silverlight environment hosted within the browser

Development Prerequisites

To start creating Silverlight applications, you need to make sure that your development environment is setup correctly. This chapter describes the setup procedure recommended by BlueFinity.

Software Requirements

The following application development software is required:

- **mv.NET Client Interface Developer**
- **Visual Studio 2010 (Professional Edition minimum)**
- **Silverlight 4 Tools package**. Download from http://www.microsoft.com/downloads (search for "Silverlight 4 Tools")
- **Silverlight 4 Toolkit**. Download from http://silverlight.codeplex.com/releases

Solution Objects Entity Model

When using mv.NET, all data access within the browser resident section of the Silverlight application is done via a Solution Objects generated business objects layer. Thus, a prerequisite for using Silverlight with mv.NET is that you must use mv.NET's Solution Objects component to create a business objects layer.

Because there will be 2 projects in your Silverlight Visual Studio solution – one relating to the software that runs inside the browser and one that is the web site hosting your web service – you need to generate 2 different flavors of your business object layer. The Business Access Layer definition form within the Entity Modeling section of the Data Manager allows you to specify the target runtime environment. Please refer to the Business Access Layer chapter of Solution Objects Developer Guide for more details on this topic.

Creating Your First Silverlight Application

Visual Studio makes creating a Silverlight application very easy. This chapter steps you through the process. Note also that mv.NET's XAML Wizard can also be used to create a Visual Studio solution for you automatically.

Create a New Solution

Firstly, within Visual Studio 2010 you need to create a new solution/project based on the "Silverlight Application" template. This template will create a solution containing 2 projects as discussed in the previous chapter.

Add Assembly References

Both projects in your new solution need to reference several assemblies in order to have access to a range of important classes.

Web Site Project References

Within your web site project, you need to add references to the following assemblies:

- BlueFinity.mvNET.SolutionObjects.dll
  (the above assembly can be found in Program Files|BlueFinity|mv.NET|Version4.0|bin|Public Assemblies)
- the Winform/Webform runtime environment version of your BAL
  (see section Solution Objects Entity Model)

Browser Project References

Within your browser project you need to add references to the following assemblies:

- BlueFinity.mvNET.CommonSL.dll
- BlueFinity.mvNET.CoreObjectsDALSL.dll
- BlueFinity.mvNET.CoreObjectsSL.dll
- BlueFinity.mvNET.SilverTools.dll
- BlueFinity.mvNET.SolutionObjectsSL.dll
  (the 5 assemblies above can be found in Program Files|BlueFinity|mv.NET|Version4.0|bin|Silverlight)
- System.Windows.Controls.dll
- System.Windows.Controls.Input.dll
- System.Windows.Controls.Data.dll
- System.Windows.Controls.Data.Input.dll
- System.Windows.Controls.Navigation.dll
- System.Windows.Data.dll
  (the 6 assemblies above can be found in C:|Program Files|Microsoft SDKs|Silverlight|v4.0|Libraries|Client
- System.Windows.Controls.Input.Toolkit.dll
- System.Windows.Controls.Layout.Toolkit.dll
- System.Windows.Controls.Toolkit.dll
  (the 3 assemblies above can be found in C:|Program Files (x86)|Microsoft SDKs|Silverlight|v4.0|Toolkit|Apr10|Bin
- the Silverlight runtime environment version of your BAL
  (see section Solution Objects Entity Model)

Create a Web Service

Within the web site project, you need to create a web service - this will be the point of contact for the browser project to gain access to database and other non-browser resident resources.

To create a new web service right-click the web site project name within the Visual Studio Solution Explorer and select the "Add ► New Item" option. From the resulting list, select "Silverlight-enabled WCF Service". Change the name of the new service to a meaningful name, e.g. for the SOP example application the service name has been called SOPBALService to reflect the fact that this service is going to provide services for the SOP business access layer running inside the browser.

Once the service has been created by Visual Studio, right-click the "xxx.svc" entry that will now exist within the Solution Explorer listing for the project and select

the "View Code" option. Visual Studio will have created a stub entry for service – you need to amend this template code as follows:

1. Add "using" (C#) or "Imports" (VB) statements for the following namespaces at the top of the code module:
   - BlueFinity.mvNET.SolutionObjects
   - the namespace of your entity model

2. Replace the default "DoWork" template-supplied web method with the Solution Objects "Action" web method. This web method can be found in the "Code Snippets" tab of the XAML Wizard. Please refer to the XAML Wizard chapter of this guide for further details. A copy of this template is given below:

(VB)

```
<OperationContract()>
Public Sub Action(ByRef ActionGUID As String, ByRef ActionType As String, ByRef
                  EntityName As String, ByRef RepositoryDetails As String, ByRef
                  Param1 As String, ByRef Param2 As String, ByRef Param3 As String,
                  ByRef Param4 As String, ByRef Param5 As String, ByRef Param6 As
                  String, ByRef Param7 As String, ByRef DatasourceData As String,
                  ByRef ErrorDescription As String)

    Try
        Repository.ProcessAction(ActionGUID, ActionType, EntityName,
                                 RepositoryDetails, Param1, Param2, Param3,
                                 Param4, Param5, Param6, Param7, DatasourceData,
                                 ErrorDescription)
        Catch ex As Exception
            ErrorDescription = ex.Message
    End Try

End Sub
```

(C#)

```
[OperationContract()]
public void Action(ref string ActionGUID, ref string ActionType, ref string
                   EntityName, ref string RepositoryDetails, ref string Param1, ref
                   string Param2, ref string Param3, ref string Param4, ref string
                   Param5, ref string Param6, ref string Param7, ref string
                   DatasourceData, ref string ErrorDescription)
{
    try
    {
        Repository.ProcessAction(ActionGUID, ActionType, EntityName,
                                 RepositoryDetails, Param1, Param2, Param3, Param4,
                                 Param5, Param6, Param7, ref DatasourceData, ref
                                 ErrorDescription);
    }
    catch (Exception ex)
    {
        ErrorDescription = ex.Message;
    }
}
```

Create a Web Service Reference

In order to use the web service that you have just created within the web site project a reference to it must be created within the browser project.

Before performing this step, it is always good practice to perform a full Solution Rebuild within Visual Studio.

To add a web service reference, right-click the browser project node within the Server Explorer and select the "Add Service Reference" option.

In the resulting window, click the "Discover" button and when the name of your web service's svc file appears in the lower window double-click its name to reveal the service entry directly beneath the svc entry. When the service name is displayed click it once and then enter the namespace name at the foot of the window. We recommend using a standard of <web service name>Proxy - for example, SOPBALServiceProxy. Click the "OK" button when you have entered the required namespace name.

The XAML Wizard

mv.NET's XAML Wizard has an option to create a Visual Studio application with all of the above infrastructure already configured. Please refer to the XAML Wizard chapter for further details on this.

Summary

So, at this point we have a Silverlight application with a Solution Objects compliant web service hosted within a web site and referenced by a Silverlight browser component project along with the necessary assembly references. We are now ready to start creating the content of our application.

Deploying Silverlight Applications

Once you have developed your Silverlight application you will need to deploy it to either your test or production web server. This chapter covers several mv.NET specific issues relating to this subject area.

Silverlight Application Deployment Overview

Deploying a Silverlight application is very similar to deploying a normal web site application. There is much on-line documentation and user-forum postings on this subject area – some links to the ones we have found useful are listed below.

http://www.silverlight.net/learn/whitepapers (Getting Started -> Silverlight Enterprise Deployment)
http://msdn.microsoft.com/en-us/library/ff921170(PandP.20).aspx
http://learn.iis.net/page.aspx/262/configuring-iis-for-silverlight-applications

mv.NET Specific Issues

Although the action of deploying a Silverlight application is something which is not related per se to the use of mv.NET within the application, there are a few issues which warrant attention in this documentation.

a) If the web server does not have the mv.NET SRDK, CRDK or CID installed, you will need to ensure that the following mv.NET dlls are included in the bin folder of the web site:

   Bluefinity.mvNET.CoreObjectsDAL.dll

Bluefinity.mvNET.Common.dll
Bluefinity.mvNET.License.dll
Bluefinity.mvNET.mvNETDB.dll
Bluefinity.mvNET.GatewayClient.dll

b) If the web server is not hosting the mv.NET Session Manager you will need to ensure that the address of the Session Manager is supplied in the DataRepository initialization that will be in the VB/C# code behind of your application forms. Please refer to the "Initializing Data Access" section of the "Reading/Selecting Data" chapter of the Solution Objects Developer Guide for details on how this can be done. An example of this would be as follows:

(VB)

```
Dim connDetails As String = "Server=SOP|SessionManagerAddress=192.1.200.98"
SOPdata = Repository.Initialize(connDetails, ServiceProxy, LayoutRoot)
```

(C#)

```
String connDetails = "Server=SOP|SessionManagerAddress=192.1.200.98";
SOPdata = Repository.Initialize(connDetails, ServiceProxy, LayoutRoot);
```

An alternative to supplying the Session Manager address in the DataRepository initialization would be to create a ConfigurationPath file to specify the location of the Session Manager. Please refer to the "Configuration Database" section of the "Data Manager" chapter of the Core Objects Developer Guide for details on how this can be done.

The XAML Wizard

To speed up your Silverlight application development process BlueFinity has provided a utility which allows you to quickly and easily define the structure of your application menu and navigation scheme, along with the content of data maintenance forms. This utility is called the "XAML Wizard" and its features are described in this chapter.

XAML Wizard Feature Summary

The XAML Wizard provides the following features:

- Allows an application menu hierarchy to be defined
  - menu options can be hidden/displayed based on user privilege level
  - images can be optionally associated with each menu option
  - menus can be displayed as either cascading drop-down menus or side bar "Outlook" style
- Allows application data entry forms to be defined
  - Properties from entities within an entity model can be dragged and dropped onto a design surface
  - The designer detects the types of controls relevant for a property based on property definition
  - Form content is created in a screen resolution independent manner
- XAML markup and code behind (VB or C#) to implement the menu and form designs is generated into the required application folder

The principle of the XAML wizard is to allow you to get your application development process "off the ground" quickly and easily by performing a lot of the initial ground work automatically. The code generated by the XAML wizard can be enhanced as required in a completely free manner using the standard Visual Studio tools. You are not locked into using only the features supported by the

wizard. This, in effect, gives you the best of both worlds – rapid application development with no limits!

Invoking the XAML Wizard

The XAML wizard is invoked in one of 2 ways depending on which version of the Data Manager is being used.

If the standalone Data Manager is being used, the XAML Wizard is invoked using the "XAML Wizard ► Open" menu option at the top of the Data Manager window.

If the Visual Studio add-in version of the Data Manager is being used, the XAML Wizard is invoked by clicking the XAML Wizard icon at the top of the Data Manager tool window:

On invoking the XAML Wizard, you will be prompted to open an existing form design or create a new one.

Creating Form Designs

The form designer within the XAML Wizard allows you create sophisticated data maintenance forms. To create a new form design, click the "Form Design" tab within the Wizard's main area then select the "File ► Create New Design" menu option from the Wizard's top menu bar.

A window prompting for 3 fields of information is then displayed:

Location of data access layer: The form design process is driven largely by the content of an entity model. This entity model will have been previously produced using mv.NET's Solution Objects component. As part of the data access layer creation process, Visual Studio will generate an xml file in addition to the dll assembly. It is this xml file that you need to select. The XAML Wizard inspects the xml file content and obtains all the necessary information for it to understand the details of the entity model for intelligent form designing.

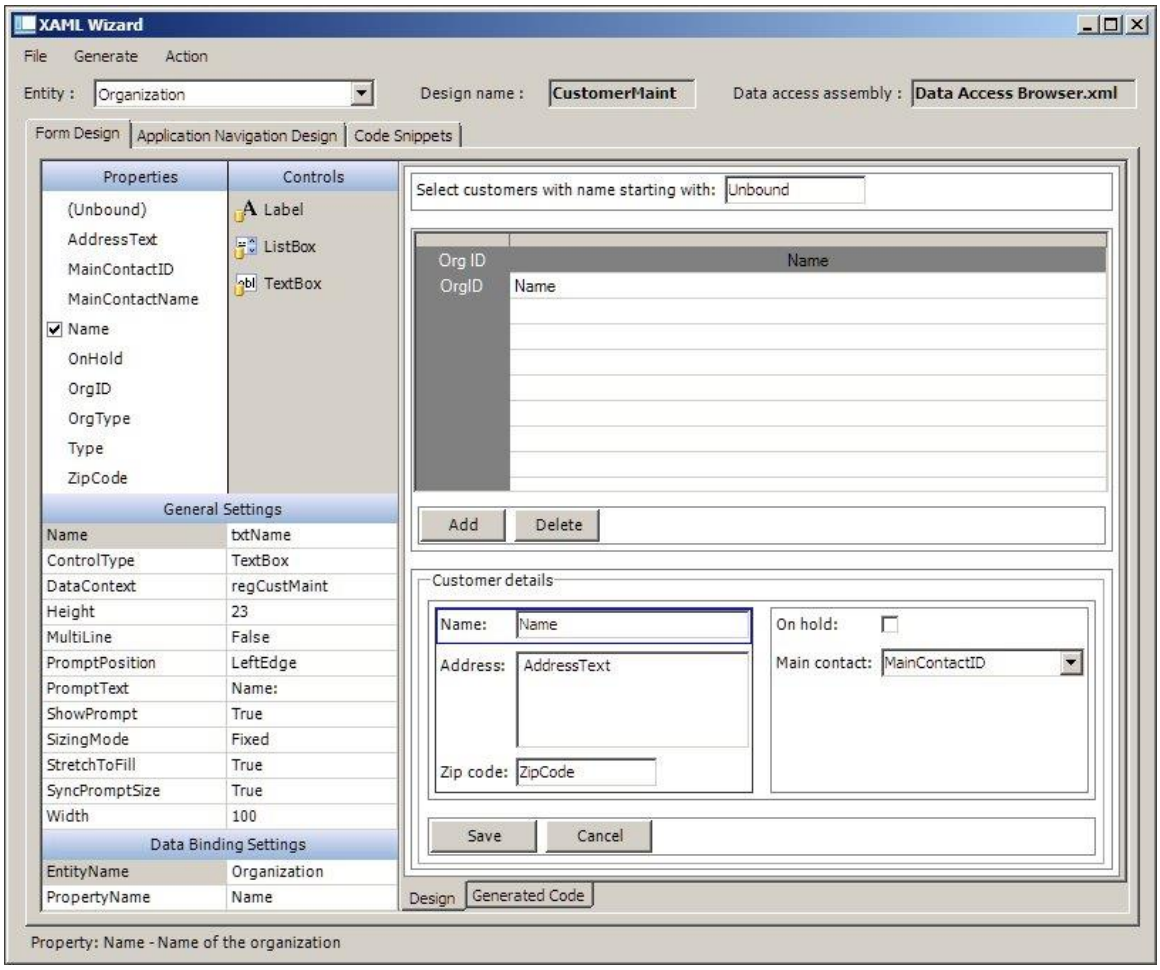
Name of new design: This is the name of the form that will be used within XAML and code-behind generated code. It should not contain spaces or punctuation characters.

Location to save design to: The path of the folder into which the form design file will be saved.

On clicking the Accept button, a new blank design surface will be displayed.

Form Designer Layout

The appearance of the XAML Wizard's form designer is shown in the following screenshot:



At the top of the screen is shown the form design name and associated data access layer xml file. Also, in the very top left corner there is a combo box listing all of the entity names within the data access layer.

The "Form Design" tab is divided into 5 main areas:

Properties : This area lists all of the properties contained in the currently selected entity. A tick is displayed alongside the currently selected property name(s). To select more than one property name hold down the Ctrl key whilst clicking a property name. The name "(Unbound)" is always shown as the first entry in the Properties list – more on this later.

Controls : When a property name is selected, the list of controls that are capable of having the property's value bound to them is listed in the Controls area.

Form design surface : The right half of the form designer screen is used to display the form design itself. It is important to recognize the fact that this is a logical display of the form content and is only an approximation of what the form will look like when displayed inside a web browser.

The purpose of the form designer is to allow you to make sure that all controls are positioned correctly relative to each other and that the necessary data binding characteristics are defined.

The form design surface area is a tabbed display. The "Design" tab contains the form design appearance. The "Generated Code" tab allows you to view the code set that was last generated based on the form design.

General Settings : When an element of the form design is selected in the design surface display its general characteristics are displayed in the General Settings grid. The exact content of the General Settings will vary depending on what kind of element is selected. All possible entries are listed in the "General Settings" section below.

Data Binding Settings : In addition to the General Settings, if relevant, the data binding characteristics of the selected form design element are displayed in the Data Bindings Settings grid. As with the General Settings, the exact content of the Data Bindings Settings will vary depending on what kind of element is selected. All possible entries are listed in the "Data Bindings Settings" section below.

Form Designer Regions

The XAML Wizard form designer is based around the concept of dividing a form into several logical "regions". A region can contain controls (text boxes, check boxes, labels etc.) and other (nested) regions. Regions form the basis of the form designer's ability to generate screen resolution independent XAML code and its ability to bind different sections of a form to different data contexts.

When a new form design is created a region called, by default, "regionRoot" is automatically created. This is the container that will contain all other regions and controls on the form. It is not usual, although possible, for the root region to directly contain controls. It is more normal for the root region to contain several nested (child) regions, with these child regions containing controls and further nested regions as necessary.

You can create a nested region within another region in one of 2 ways:

1. Right-click the surface of the parent region and select the "Insert New Nested Region" option from the context menu.
2. Select the parent region and select the "Insert New Nested Region" option from the "Action" menu at the very top of the XAML Wizard screen.

Sometimes, if a region contains many controls and/or other nested regions, it is difficult to click on its surface directly; therefore, if you right click any element

within a form design you can choose the "Select Parent Region" option to get its parent region selected within the form designer. The "Select Parent Region" menu option is also contained within the "Action" menu at the very top of the XAML Wizard screen.

A region can organize (arrange) its content (controls and other nested regions) either vertically (one above the other) or horizontally (side-by-side). This behavior can be configured using the "OrganizationDirection" general setting for a region. The organization direction of a region is one of the main ways of controlling the relative positioning of form content.

In the screenshot shown in the previous section, the form design is divided into 3 main regions. The top region holds the control that prompts for search criteria. The middle region holds the list of selected customers along with the Add and Delete buttons. The 3rd, bottom region holds all the elements used to maintain the details of the currently selected customer.

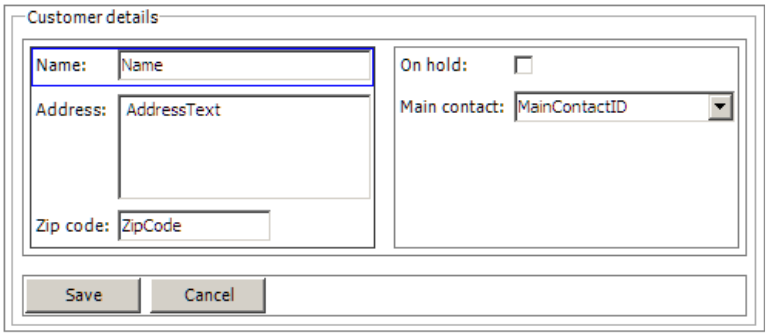
In order to see the fine detail of how this example form is composed, let's zoom into the detail of each main region.

The first thing to note is that the root region's "OrganizationDirection" general setting is set to "Vertical" – this is its default setting.

The top main region is quite simple; it only contains a single unbound TextBox control that allows us the capture keystrokes from the user.

The middle main region is, again, quite simple; it contains a DataGrid control that is used to display the list of selected customers and it also contains a nested region which is used to hold the Add and Remove buttons. Because the OrganizationDirection of the middle region is set to "Vertical", the nested region appears beneath the DataGrid. The nested region contains 2 buttons, Add and Remove. These have been added by dragging and dropping an "AddRemove" control from the list of controls displayed when the "(Unbound)" property entry is selected. The OrganizationDirection of the nested region has been set to "Horizontal" – which is why the 2 buttons appears side-by-side as opposed to one above the other.

The bottom main region is the most complex. It is shown below:



The first thing to notice about this region is the fact that it is being rendered as a GroupBox – i.e. it has a visible surrounding border with a top-left text caption. This is because its "ContainerType" general setting has been set to "GroupBox".

The bottom region contains 2 nested regions, organized vertically. The first of these nested regions contains a further 2 nested regions and because its OrganizationDirection is set to "Horizontal", these 2 nested regions appear side-by-side. Each of the 2 side-by-side regions are organized vertically and contain a series of controls that appear one above the other.

The lower nested region within the bottom region contains 2 buttons that have been added by dragging and dropping a "SaveCancel" control from the list of controls displayed when the "(Unbound)" property entry is selected. Its OrganizationDirection has been set to "Horizontal" – which is why the 2 buttons appears side-by-side.

It can therefore be seen that by using a combination of nested regions in conjunction with vertical/horizontal region content organization a flexible form design that behaves intelligently when screen sizes change can be easily and quickly created.

Re-arranging Region Content

The content of a region can be altered in several different ways.

Firstly, as already discussed, the OrganizationDirection general setting can be set to control the basic stacking of content.

Secondly, the logical ordering of region content can be altered by using either the right-click context or Action menus to access the Move Forward/Backwards options. These allow the currently selected element to be moved relative to its neighboring elements.

Thirdly, controls can be dragged and dropped to change either their relative position within the same region or moved into a different parent region. During

control drag/drop blue lines are displayed at the edges of existing form controls to indicate the target position of the new control.

Entire regions cannot be dragged and dropped. If you wish to move a region you can use the menu options contained in either the right-click context or Action menus.

Adding Controls to Regions

Any region may contain controls. "Control" is the generic term used to denote an input field or any user-interactive widget.

From the XAML Wizard's perspective, there are 2 fundamentally different types of controls – "bound" and "unbound".

A "bound" control is one which is associated with one or more properties of an entity. An "unbound" control has no direct association with an entity property.

A bound control is created as follows. Firstly, select the relevant entity name from the top-left Entity combo box. Next, select the required property name from the list of names displayed in the Properties area. If you wish to create a control that can display more than one property (e.g. a DataGrid) you need to hold down the Ctrl key whilst clicking a series of property names.

Once the relevant property name(s) have been selected, you can drag and drop one of the control icons listed in the Controls area onto the form design surface. The controls listed within the Controls area will change depending on the data characteristics of the selected property.

During control drag/drop blue lines are displayed at the edges of existing form controls to indicate the target position of the new control. If you drag/drop a control onto a DataGrid, you also have the additional option of creating a new column (a green line is used to indicate where the new column will be created).

An unbound control is created by selecting the "(Unbound)" entry within the Properties area. It does not matter which entity name is selected – the "(Unbound)" entry is always displayed at the top of the Properties list. The names of available unbound controls are listed in the Controls area from which you can drag and drop onto the form design surface as required.

General Settings

The General Setting grid contains several rows allowing you to maintain a range of values that will ultimately control the code that is generated by the XAML wizard. The entries shown in the General Settings grid will vary according to the type of element selected within the form design surface.

The height of the General Settings grid may be adjusted by dragging its title bar up and down. Double-clicking its title bar will toggle it between a collapsed and visible state.

Below are tables explaining all of the possible settings that may be displayed within the General Settings grid.

Both Region and Control element types:

Setting Name	Description
Name	The name of the element. This name will be used in generated XAML and code-behind. It should not contain spaces or punctuation characters.
DataContext	The name of the element which is to provide data at runtime for this element (and potentially all of its child elements).
DataContextProperty	The name of the property (based on the value of the DataContext setting) which is to provide data at runtime for this element.
Margins	This allows you to override the default spacing between elements on a form.
SizingMode	The way in which the size of the element is to be determined. May be one of: **Fixed** – the element will be assigned a fixed (pixel) size. **Percentage** – the element's size will be set as a percentage of the overall space available within the parent region. **Auto** – (Regions only) the region's size will be based upon the size requirements of its child elements.
FixedSize	The (pixel) size of the element – only relevant if SizingMode set to Fixed.
PercentageSize	The percentage size of the element – only relevant if SizingMode set to Percentage.

Regions only:

Setting Name	Description
ContainerType	The type of container that the region is to be rendered as. May be one of Panel, GroupBox or Tabbed.
OrganizationDirection	The direction in which region content will be stacked. May be one of Vertical or Horizontal.
Heading	(GroupBox regions only) The text to appear in the top-left corner of the region.
TabEarPosition	(Tabbed regions only) The position of tab ears (Top or Bottoms).
TabEarText	(Tabbed regions only) The text to appear in the currently selected tab ear of the region.
ValidationBindings	Allows the list of controls whose enabled/visible state is to be automatically adjusted in step with the validation status of DataContext of the region.

Controls only:

Setting Name	Description
CaseConversion	(TextSelect and AutoCompleteBox controls only) Indicates how the text that the user has entered is to be converted before being passed into the specified selection method.

<table>
<tr><td>ColumnSettings</td><td>(DataGrid controls only) This entry contains a number of sub-settings all relating to the currently selected grid column. A grid column is selected by clicking it within the form designer.
<table>
<tr><th>Sub-setting Name</th><th>Description</th></tr>
<tr><td>BodyTextAlign</td><td>The alignment style of the content of the column. May be one of Left, Center or Right.</td></tr>
<tr><td>HeaderTextAlign</td><td>The alignment style of the column header. May be one of Left, Center or Right.</td></tr>
<tr><td>Sizing</td><td>Controls how the width of the column is determined. May be one of Fixed or Percentage</td></tr>
<tr><td>Width</td><td>The fixed width (in pixels) of the column.</td></tr>
<tr><td>WidthPercent</td><td>(Percentage sized columns only) The percentage width of the column relative to the overall width of the grid.</td></tr>
</table></td></tr>
<tr><td>CustomXAML</td><td>Allows a string of custom XAML attributes to be entered for a control. The content of this property will be inserted into the generated code.</td></tr>
<tr><td>FontSettings</td><td>Allows the default font characteristics to be overridden.</td></tr>
<tr><td>Height</td><td>The height (in pixels) of the control if its height is not being calculated automatically.</td></tr>
<tr><td>LinkedTo</td><td>(TextSelect controls only) Indicates the control to which this TextSelect instance is to pass the list of selected items.</td></tr>
<tr><td>ListEntity</td><td>(AutoCompleteBox controls only) The entity type to provide the data displayed in the drop down portion of the control.</td></tr>
<tr><td>MinimumSize</td><td>The minimum size (in pixels) of the control if its height is not being calculated automatically.</td></tr>
<tr><td>MinimumPopulateDelay</td><td>(AutoCompleteBox controls only) The control will wait until no user input occurs for this period of time (milliseconds) before re-populating the list of matches in its drop-down section.</td></tr>
</table>

MinimumPrefixLength	(AutoCompleteBox controls only) Selection processing will only be invoked if the length of user input text is equal or greater than this value.
MultiLine	(TextBox controls only) Indicates whether the TextBox is to display more than one line of information.
PageSize	(DataGrid controls only) A non-zero value for this property will result in a standard Silverlight DataPager control being used to paginate the data source assigned to the DataGrid. The DataPager will be displayed beneath the DataGrid control.
PromptPosition	Indicates the position of the associated prompt text.
PromptText	The text to be displayed alongside the control.
SelectionMethod	(TextSelect and AutoCompleteBox controls only) The selection method to be used by the control. The method names listed here are single argument static selection methods of the related entity.
ShowPrompt	Indicates whether a prompt is to be displayed alongside the control.
StretchToFill	Indicates whether the control's height or width is to be extended to fill the available space within the parent region.
SyncPromptSize	Indicates whether the length of prompt text is to be used to determine how the left edges of controls are to be aligned within a region.
TextAlignment	(TextBox and Label controls only) Allows the alignment of the text containing within the control to be set.

Data Binding Settings

The Data Bindings Settings grid contains several rows allowing you to maintain a range of values that will govern the way in which the runtime content of a control is mapped to the underlying entity model. The entries shown in the Data Bindings Settings grid will vary according to the type of element selected within the form design surface.

The height of the Data Binding Settings grid may be adjusted by dragging its title bar up and down. Double-clicking its title bar will toggle it between a collapsed and visible state.

Below is a table explaining all of the possible settings that may be displayed within the Data Binding Settings grid. Data bindings are only relevant for Controls.

<table>
<tr><th>Setting Name</th><th>Description</th></tr>
<tr><td>EntityName</td><td>(Read-only) The name of the entity to which this control is bound.</td></tr>
<tr><td>PropertyName</td><td>The name of the entity property to which this control is bound.</td></tr>
<tr><td>EditingControl</td><td>(DataGrid controls only) The type of editor to be used in the currently selected grid column.</td></tr>
<tr><td>DropdownSettings</td><td>(ComboBox controls/columns only) This entry contains a number of sub-settings all relating to the data which is to be displayed in the dropdown section of a ComboBox control or the column of a DataGrid which has its EditingControl setting set to ComboBox.
<table>
<tr><th>Sub-setting Name</th><th>Description</th></tr>
<tr><td>EntityType</td><td>(Read-only) The name of the entity to which the dropdown data is bound.</td></tr>
<tr><td>InitialDisplayProperty</td><td>The name of a Property (from the main bound entity) whose value is to be used as the initial value displayed in the ComboBox control. This will typically be a calculated property based on the main PropertyName setting. By specifying a value for this setting, the initial value displayed within the ComboBox can be obtained without the need to assemble the list of entity instances that will provide the dropdown values.</td></tr>
<tr><td>EditingControl</td><td>(DataGrid controls only) The type of editor to be used in the currently selected grid column.</td></tr>
<tr><td>ItemsSource</td><td>The name of a Property (from the main bound entity) whose value is to be used to provide the dropdown data items.</td></tr>
<tr><td>DisplayProperties</td><td>The name(s) of the properties (from the entity specified in the EntityType property above) to be displayed in the dropdown portion of the ComboBox.</td></tr>
</table>
</td></tr>
</table>

		The value of this setting is set using the builder button shown at the right edge of this setting's grid row.
	ValueProperty	The name of the property (from the entity specified in the EntityType property above) which maps onto the foreign key specified in the main PropertyName setting. This will typically be the primary key property of the entity specified in the EntityType property above.
InitialDisplayProperty	(AutoCompleteBox controls only) The name of a Property (from the main bound entity) whose value is to be used as the initial value displayed in the control. This will typically be a calculated property based on the main PropertyName setting. By specifying a value for this setting, the initial value displayed within the control can be obtained without the need to assemble the list of entity instances that will provide the dropdown values.	
ValueProperty	(AutoCompleteBox controls only) The name of the property (from the entity specified by the ListEntity property) which maps onto the foreign key held in the property specified by the PropertyName setting above. This will typically be the primary key property of the entity specified in the ListEntity property.	

The Application Navigation Designer

The application navigation designer within the XAML Wizard allows you create the menu structure used by end-users to navigate around the functionality provided by an application.

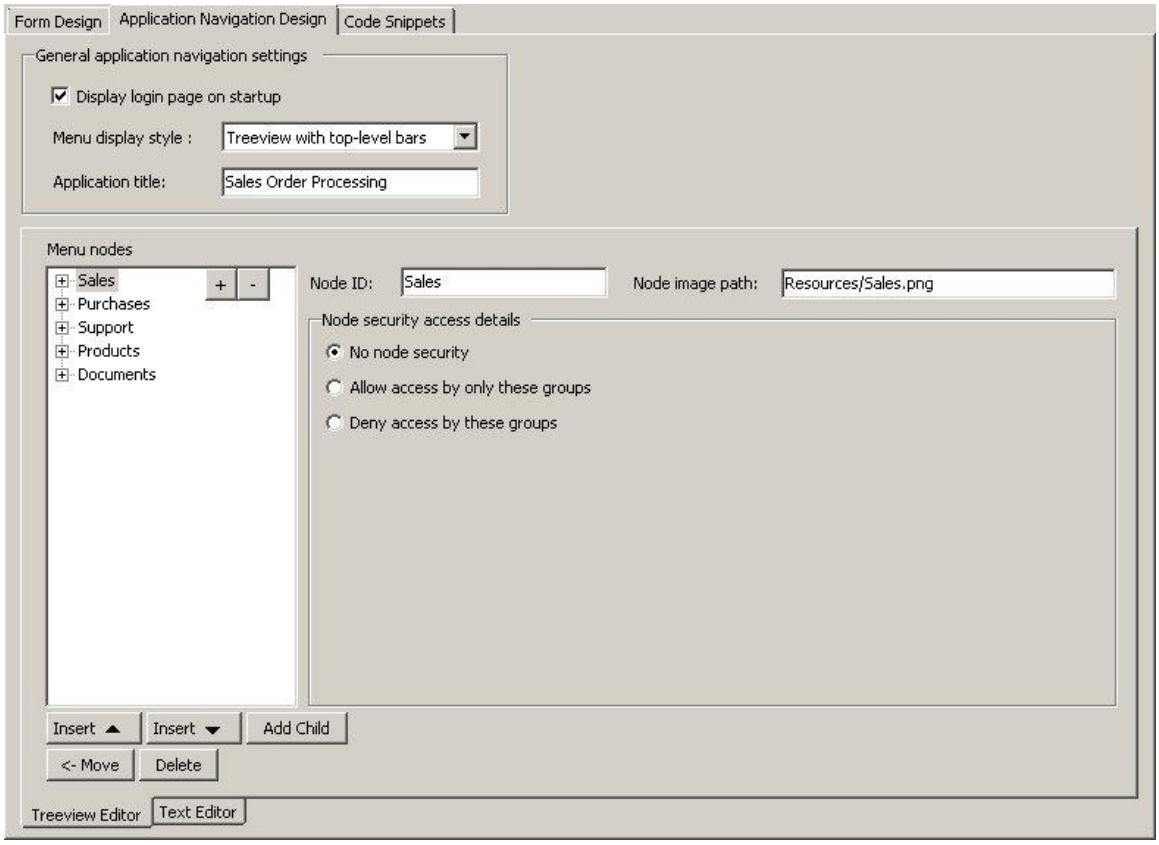
To create a new navigation design, click the "Application Navigation Design" tab within the Wizard's main area then select the "File ► Create New Design" menu option from the Wizard's top menu bar.

A window prompting for 3 fields of information as per Creating Form Designs is then displayed.

On clicking the Accept button, a new blank design surface will be displayed.

Application Navigation Designer Layout

The appearance of the XAML Wizard's application navigation designer is shown in the following screenshot:



The top section of the navigation designer allows some general settings to be maintained:

Display login page on startup : This check box allows you to indicate whether a user name/password entry dialog needs to be display on first invocation of the application's home page. This allows you to capture and authorize user credentials. Note, this is different to user authentication which will be typically handled using one of several standard web application authentication techniques – a subject matter which is beyond the scope of this guide but one about which there is much on-line content.

Menu display style : This combo box allows you to indicate the style of menu display required for the application. There are 2 options currently available:

- *Cascading menus*
  This option results in a series of cascading menus and sub-menus being displayed at the top of the page.
- *Treeview with top-level bars*

This option results in a side-bar navigation area being displayed at the left edge of the page – a navigation style introduced and made popular by Microsoft's Outlook product.

Application title : The text to be displayed at the top of the main page.

The lower section of the application navigation designer allows you to define the menu hierarchy. There are 2 ways to define the menu hierarchy; a treeview based designer and a text-editor based designer.

Treeview-based Application Navigation Design

The "Treeview Editor" tab on the application navigation designer's tab allows you to view and edit menu hierarchy using a treeview control. The treeview control provides a nice visual presentation of the hierarchy with the ability to collapse nodes as required. At the top of the treeview control are 2 buttons that allow the entire menu hierarchy to be either expanded or collapsed.

As a menu node is selected, its current definition is displayed to the right of the Treeview:

Node ID : This allows a unique identification string to be associated with the node. This ID is used by code behind within the application to identify when a specific menu node is selected.

Node image path : For "Treeview with top-level bars" menu designs, this option allows you to specify the relative path of the file which contains an image to be displayed at the left side of a menu option.

Node security access details : This input area allows you to define the security details for a node, i.e. it allows you to define which group(s) of users are allowed to see/access the menu option.

Underneath the treeview control are a series of buttons that allow you to insert/remove new menu nodes into/from the treeview.

Text Editor-based Application Navigation Design

The "Text Editor" tab on the application navigation designer's tab allows you to view and edit menu hierarchy using a textbox control. A screenshot of an example is shown below:

```
Sales   ID="Sales" Image="Resources/Sales.png" Allow=""
→   Customer Details   ID="CustomerDetails" Allow="MANAGER ENGINEER"
→   →   General Details   ID="CustomerGeneral"
→   →   Contacts   ID="CustomerContacts"
→   Quotations   ID="SalesQuotations" Image="" Allow=""
→   Orders   ID="SalesOrders"
→   Projects   ID="SalesProjects"
→   Invoices   ID="SalesInvoices"
→   Credits   ID="SalesCredits"
→   Payments   ID="SalesPayments"
→   Reporting   ID="SalesReporting" Image="Resources/Graph.png"
Purchases   ID="Purchases" Image="Resources/Purchases.png"
→   Supplier Details   ID="SupplierDetails" Allow="MANAGER SECRETARY"
→   →   General Details   ID="SupplierGeneral"
→   →   Contacts   ID="SupplierContacts"
→   Current Requirements   ID="PurchaseRequirements"
→   Orders   ID="PurchaseOrders"
→   Invoices   ID="PurchaseInvoices"
→   Credits   ID="PurchaseCredits"
→   Payments   ID="PurchasePayments"
→   Reporting   ID="PurchaseReporting"
Support   ID="Support" Image="Resources/Person.png"
→   In-progress   ID="SupportInProgress"
→   Fixed Waiting Close   ID="SupportWaitingClose"
→   Closed   ID="SupportClosed"
→   Reporting   ID="SupportReporting"
Products   ID="Product" Image="Resources/Products.png"
→   Catalog   ID="ProductsCatalog"
→   Equipment Register   ID="ProductsEquipment"
→   Stock Take   ID="ProductsStockTake"
→   Reporting   ID="ProductsReporting"
Documents   ID="Documents" Image="Resources/Documents.png" Deny="MANAGER"
→   General   ID="DocumentGenerals"
→   Transaction-related   ID="DocumentsTransaction"
```

The right-pointing arrows denote tab characters which are used to indicate node indentation. Each node's text and definition is held on a single line as illustrated above.

The text editor provides a convenient way to quickly enter a menu hierarchy, with the ability to cut and paste menu node(s).

Code Snippets

The third main tab on the XAML Wizard's screen ("Code Snippets") allows you to access a library of useful code snippets. You are able to add your own snippets this library if required.

Code snippets are located in the following folder:

```
C:\Program Files\BlueFinity\mv.NET\Version4.0\Code Templates\WPF\XAML\Code Snippets
```

And

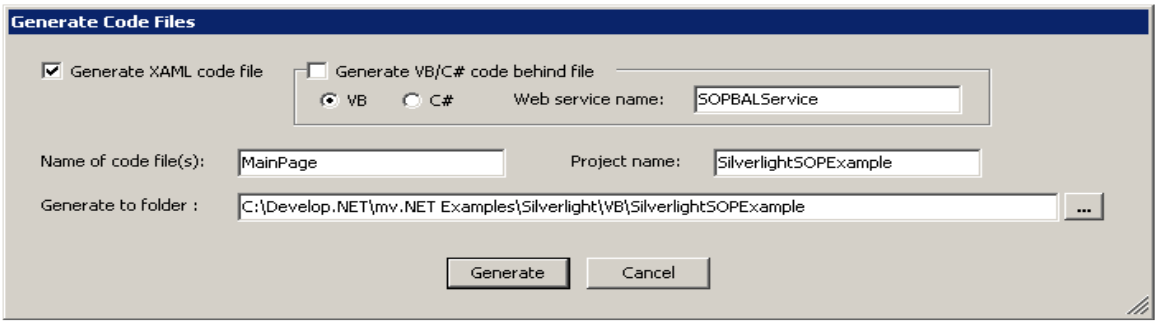
```
C:\Program Files\BlueFinity\mv.NET\Version4.0\Code Templates\WPF\Code Behind\Code Snippets
```

Each snippet is a simple text file with the first line holding a description of the snippet and the 2nd and subsequent lines holding the snippet content.

Code Generation

After creating either your form or application navigation design you can get the XAML Wizard to generate the relevant XAML and/or VB/C# code behind for the design. This code can then be included within your Silverlight application.

Code generation is invoked by selecting the "Generate ► Full Code Files" menu option from the Wizard's top menu bar. On selecting this option, the following form is displayed:



The top section allows you to control which types of code are generated. For the code behind section you need to specify the name of the web service that has been created in the web site project of your Visual Studio solution – see section Create a Web Service.

The input fields in the lower half of the screen are explained below:

Name of code file(s) : This is the name of the XAML/code-behind file that is to be generated. This/these files are the ones that will need to be included within your Visual Studio project.

Project name : This needs to be the name of the Visual Studio project that is to include the generated code files.

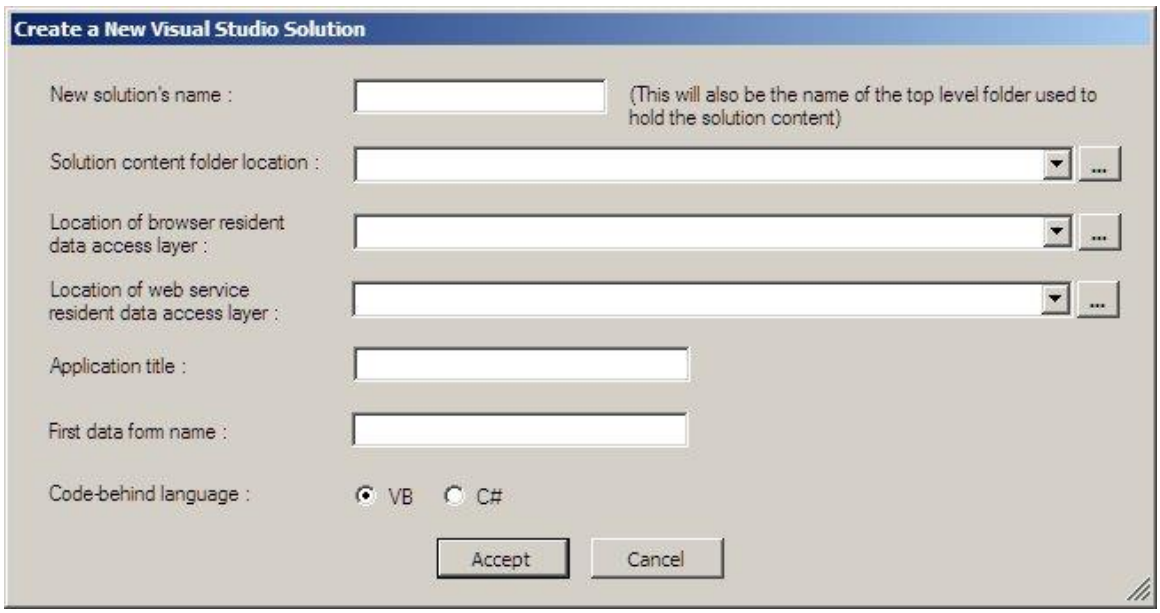
Generate to folder : This needs to be set to the relevant Visual Studio project folder.

The source code is generated when you click the "Generate" button.

Creating a New Visual Studio Solution

The XAML Wizard provides a feature which allows you to generate a complete Visual Studio solution to act as the starting point for your Visual Studio based Silverlight development activity.

To access this feature, select the File ► Create New Visual Studio Solution menu option from the Wizard's top menu bar. On selecting this option, the following form is displayed:



The input fields contained on this form are explained below:

New solution's name : Enter the name of the new solution. This must not contain spaces or punctuation characters because it will be used as the programmatic name of the solution in Visual Studio.

Solution content folder location : This is the path of the folder which is to contain the new folder created to contain the content of the new solution.

Location of browser resident data access layer : This is the path of the xml file generated during the Visual Studio build of the Solution Objects entity model

generated code. It should be a Business Access Layer targeted at the Silverlight environment.

Location of web service resident data access layer : This is the path of the xml file generated during the Visual Studio build of the Solution Objects entity model generated code. It should be a Business Access Layer targeted at the Winform/Webform environment.

Application title : This needs to be a short description of the application, for example "Sales Order Processing". It will be used in the menu system generated by this process.

First data form name : This needs to be the name of your first data maintenance form, for example "CustomerMaint". This must not contain spaces or punctuation characters because it will be used as the programmatic name of the form in Visual Studio.

Code-behind language : The .NET language of the generated solution.

General Notes on Using Controls

This section contains general guidance notes/comments on the use of various controls supplied by the XAML Wizard.

ComboBox

A bound ComboBox control allows you to generate a list of drop-down values using a selection method exposed as a property. The assumption made here is that the ComboBox is bound to the foreign key field and that the value of this field can be adjusted by selecting a drop-down entry.

The selection list used here will be a selection method that returns a list of entity instances of the type that relates to the foreign key held in the main item.

AutoCompleteBox

The AutoCompleteBox is like a ComboBox but there are 2 main differences. Firstly, the list of drop-down values is generated using a (single argument) static selection method from the entity type to be listed in the drop-down section. This selection method is passed the value that has been typed in by the user.

Secondly, it is not possible to control the content of the drop-down section – it can only list a series of single property values from the drop-down entity type.

An unbound AutoCompleteBox does not have an initial value set and it does not update an associated entity property value when a drop-down entry is selected. Both a bound and unbound AutoCompleteBox can act as a DataContext source for a region or other form element.

Image

The Image control allows you drive image displaying on a page using entity data. Images are held in a folder/sub folder within your website and the basic idea is that the data held within one of an entity's property values (i.e. the Data Binding Settings PropertyName) is used to derive the name of the file holding the associated image.

The settings unique to the image control are as follows:

Setting Name	Description
DefaultImage	The name of the image file to be used if an associated image file is not found.
FileNamePrefix	The text to be added in front of the property value when assembling the associated image file name – see below.
FileNameSuffix	The text to be added after the property value when assembling the associated image file name – see below.
ImageFolder	The path name (relative to the root of the website folder) of the folder holding all images.
ImageType	The file extension of the images (e.g. png, jpg, etc) – see below.
StretchMode	Indicates the way in which images are to be stretched to fill the allocated image control size: **None** – No modification to the size of the image will be performed. If the image size is more than the size of the container the image will be cut to fit in the container. **Fill** – The image will be expanded to fill the region of the container. The aspect ratio (proportion of width and height) will not be maintained. **Uniform** – The image will be resized to fit the container, but the aspect ratio will be maintained, thus, there may be blank space in the container depending on the width and

	height of the image and container. **UniformToFill** – The image will be resized and will fill the container, but the aspect ratio will be maintained by trimming some portion of the image as required.

The full image file name that will be used is formed as follows:

```
{FileNamePrefix}{PropertyValue}{FileNameSuffix}.{ImageType}
```

If this file name is not found in the specified ImageFolder, the DefaultImage setting will be used as the **full** file name. If the DefaultImage setting is blank or is not found the image control content will be set to null.

Using a Business Access Layer within Silverlight

The way in which you utilize a Solution Objects generated business access layer (BAL) within the browser-resident portion of your Silverlight application is slightly different to the way in which you use one in other environments - for example within your web service. This chapter discusses these differences .

Why are Things Different in Silverlight?

The main reason why the use of a BAL is slightly different when utilized within the context of a Silverlight environment is due to the asynchronous nature of database access within Silverlight.

Whenever you need to access non-local resources (e.g. a database) from within the browser-resident part of your Silverlight application you must do this via an asynchronous calling pattern. That is, database actions are split into 2 phases - an action request phase and an action completed phase. The action request phase launches a background conversation to the associated web service (which then communicates with the database server). The action completed phase makes the result of that background conversation available to you when the response comes back from the database/web service.

Thus, for example, if you invoke a selection method of one of your entity classes, when that method returns control back to the next line of code you will not at that point in time have access to the resulting collection of objects. Instead, you will be returned a unique identifier associated with the background request that has just been raised on your behalf. When the collection of objects has been retrieved from the web service it is then made available to you via an event. Thus, the code which needs to access/manipulate the selected collection of objects needs to be invoked within that event handler.

This asynchronous calling pattern will inevitably cause you to code in a slightly different style to that in which you program in traditional synchronous environments. It's a bit strange at first but you soon get used to it. The good news, however, is that Solution Objects does a lot of background work to make it easier for you to work with this pattern.

Solution Objects and Asynchronous Calling

Given that the BAL which you created using Solution Objects has to comply with the asynchronous nature of database access within Silverlight, BlueFinity has built a good deal of extra functionality into the BAL code (and support assemblies) to make working with asynchronous database access easier.

Firstly, all of the background launching of database actions is done automatically by the BAL code.

Secondly, all of the handling of browser-to-web service communication work is handled by the BAL code. You simply invoke a BAL member and (at some point in the future) get access to the resulting BAL data. The BAL framework handles all of the serialization work required to move entity data bi-directionally across the web service link.

Thirdly, the BAL code automatically communicates directly with the WPF data binding mechanism used within a Silverlight application in order to support zero code handling of lazy data loading and property value recalculation.

An Example of Database Access

To illustrate how you write code to work with the asynchronous nature of Silverlight database access, a simple example is given below.

Let us suppose that our entity model contains a class called "Contact" and that we wish to use a static selection method defined for that class called "SelectByOrganization". This method allows us to select all contacts associated with a specified organization and present this as a collection of Contact instances.

We invoke the selection method as follows:

(VB)

```
contactSelectGUID = Contact.SelectByOrganization(orgKey, SOPData)
```

(C#)

```
contactSelectGUID = Contact.SelectByOrganization(orgKey, SOPDatay);
```

The contactSelectGUID variable has been previously declared as a form-scope String variable. The SOPData variable is our previously initialized DataRepository variable (see following section on The DataRepository Class in Silverlight).

When the execution of this line of code completes, the contactSelectGUID variable will contain a unique identifier associated with this specific invocation of the SelectByOrganization selection method. The execution of the SelectByOrganization within the web service will have been initiated on your behalf.

When the selection method has been completed on the web server and the results have been transported back to the browser, an event will be raised. This event is raised by the DataRepository instance and needs to be handled by a subroutine within your form code:

(VB)

```
Private Sub SOPData_ActionCompleted(ByVal ActionDetails As
BlueFinity.mvNET.SolutionObjects.AsyncActionDefinition, ByVal
ReturnedObject As Object)
```

(C#)

```
private void SOPdata_ActionCompleted
(BlueFinity.mvNET.SolutionObjects.AsyncActionDefinition ActionDetails,
object ReturnedObject)
```

The ActionDetails event argument provides access to the details of the action that has just completed (remember, it is perfectly possible within the Silverlight environment for you to have several asynchronous actions running in parallel). The ReturnedObject event argument contains the object holding the returned data (if relevant). So, in our example, if the selection method has executed successfully, the ReturnedObject argument will contain a Contacts instance, holding all of the relevant Contact objects.

The ActionDetails class has several properties that can help us figure out what action has just completed.

Firstly, it has a property called "GUID". This holds the unique identifier allocated to the action when it was first launched. In our case, this will be the value held within our contactSelectGUID variable.

Secondly, it has a property called "ActionType". This holds an enumeration value describing what type of action has just completed, a Select, a Read, and Update etc.

Thirdly, it has a property called EntityName. This is a string value holding the singular name of the entity type associated with the action. In our case this will be "Contact". Note, all entity classes have static property called "SingularName" which returns the singular name of the class as a string value - avoiding the need to use a literal string - which means that if the name of your class gets altered in the

future, you will know about it at design-time (as opposed to things blowing up at run-time) because your code will no longer compile.

Thus, we could identify our action in a number of different ways - below illustrates the use of the GUID property:

(VB)

```
If ActionDetails.GUID = contactSelectGUID Then
    Dim people As Contacts = DirectCast(ReturnedObject, Contacts)
End If
```

(C#)

```
if (ActionDetails.GUID == contactSelectGUID)
{
    Contacts people (Contacts)ReturnedObject;
}
```

Note, the ActionDetails event argument also has an ErrorDescription property - this will contain an empty string value if the action completed successfully, otherwise it will contain a description of the error encountered. It is, therefore, good programming practice to always check the value of the ErrorDescription property to ascertain whether an error occurred, rather than simply assuming that an action completed OK.

Calling Database Subroutines

Because of the asynchronous nature of Silverlight database communications, the way in which you call database subroutines via a Solution Objects generated business access layer (i.e. via a Subroutine Method) in a Silverlight application is slightly different to the way you do so in other runtime environments.

Below is the code that would be used to invoke a subroutine method (CallGUID is a form-wide string variable):

(VB)

```
CallGUID = Contact.TestSub(SOPData, "1", "9")
```

(C#)

```
CallGUID = Contact.TestSub(SOPData, "1", "9");
```

Below is the code that would be used (within the DataRepository ActionCompleted event handler to retrieve the subroutine method; note, here the completed subroutine action is being identified using the GUID captured during the above invokation:

(VB)

```
If ActionDetails.GUID = CallGUID Then
    With DirectCast(ReturnedObject, SubroutineData)
        Dim arg1 As Object = .Arguments("Arg1")
        Dim arg2 As Object = .Arguments("Arg2")
    End With
End If
```

(C#)

```
if (ActionDetails.GUID == CallGUID)
{
  SubroutineData subArgs = (SubroutineData)ReturnedObject;
  object arg1 = subArgs.Arguments("Arg1");
  object arg2 = subArgs.Arguments("Arg2");
}
```

The above code snippets illustrate how the subroutine method is invoked and then how the data returned by the arguments of the subroutine are retrieved via the SubroutineData class. In the entity model being used here the 2 arguments to the subroutine method have been named “Arg1” and “Arg2”.

The SubroutineData class is a specialized class supplied specifically to allow the results of calling database subroutines to be retrieved within the completed action event handler. It has the followed interface members:

Member Name	Return Data Type	Description
Arguments	Dictionary(String, Object)	The key is the name of the argument as defined in the entity model; the returned value is the value of the specified argument on return from the subroutine.
ArgumentByPosition	Object	Allows the value of a subroutine argument to be retrieved based on its physical position within the subroutine argument signature.
ReturnValue	Object	Returns the value defined as the return value of the subroutine.

The DataRepository Class in Silverlight

In non-Silverlight environments the Solution Objects DataRepository class performs a single task – that of managing the connection(s) to the underlying data source(s).

However, in Silverlight the DataRepository class performs a series of additional tasks. Please refer to the following chapter for further details on this.

Using the XAML Wizard

The basic structure of your code behind can be generated for you by the XAML Wizard (see previous chapter's section Code Generation). Typically, all that you will need to do is code the event handlers specific to your application and add the relevant code to the ActionCompleted event to access retrieved data.

Generating New Item IDs

A topic which warrants special attention is that of how new data items within your application are to be assigned item IDs (primary keys).

If an entity's data source update control definition within its DAC definition (see Solution Objects Developer Guide for more details on this topic) is set such that you are taking control over the "Creating new instances" action and the "Update existing instances" action, then the generation of new items IDs will be handled by your custom subroutine.

If the item ID of a new entity instance can be generated by browser resident logic without the need to invoke database resident code, the new item ID needs to be generated by your custom code within the BeforeCRUD method of the custom code module section relating to the entity. You will need to handle both the CRUDType.Create and the CRUDType.Update actions.

If the item ID of a new entity instance needs to be generated by database resident code, you need to utilize the subroutine assisted Auto ID mechanism of mv.NET in order for your backend subroutine to be called at the appropriate point within the item creation process. Please refer to the "File Properties" section of the Data Manager chapter within the Core Objects Developer Guide for further details on how to do this.

Using the DataRepository Class

In non-Silverlight environments the Solution Objects DataRepository class performs a single task – that of managing the connection(s) to the underlying datasource(s). However, in a Silverlight application the DataRepository class performs several additional tasks; this chapter outlines these extra features.

Constructing DataRepository Instances

Each form (page) within your application needs to construct its own DataRepository instance – as opposed to sharing a single instance throughout the application as would normally be the case in a non-Silverlight application.

The constructor of the DataRepository instance needs to be passed the root of the form's XAML visual tree (usually the base Grid element on a form) along with the web service proxy instance that each form also needs to construct.

This results in the need for the following lines of code at the start of every form:

(VB)

```
Private WebServiceProxy As New SOPBALServiceProxy.SOPBALServiceClient
Private SOPData As DataRepository

Public Sub New()

    InitializeComponent()
    InitializeDataComponents()

End Sub
```

(C#)

```
Friend Sub InitializeDataComponents()

    SOPData = Repository.Initialize(WebServiceProxy, LayoutRoot)
```

```
End Sub

private SOPBALServiceProxy.SOPBALServiceClient webServiceProxy = new
SOPBALServiceProxy.SOPBALServiceClient();
private DataRepository SOPData;

public OrderSelect()
{
    InitializeComponent();
    InitializeDataComponents();
}

internal void InitializeDataComponents()
{
          SOPData = Repository.Initialize(webServiceProxy, LayoutRoot);
}
```

If you create your data form using the XAML Wizard this code will be generated for you. Please refer to the Code Generation section in the previous chapter for further details on this.

The LoadData Method

The LoadData method should be used to set the data context of controls in preference to setting the DataContext/ItemsSource properties of controls directly. The reason behind this is that by using the LoadData method the DataRepository instance is able to perform additional data-related tasks such as a updating the data context of other related regions on the form.

The LoadData method needs to be passed 2 arguments. The first is a reference to the control which is to be assigned the new data. The second is the new data itself. This second argument can be either a singular/collective object instance obtained using your entity model classes or it can be the GUID string returned by the invocation of a select method or read. For example:

```
SOPData.LoadData(dgrOrganizations, Organization.SelectCustomersByName(
txtCustSelect.Text.ToUpper, SOPData))
```

The above statement loads the "dgrOrganizations" control (a DataGrid) with the list of customers returned by the SelectCustomersByName method of the Organization class.

DataRepository Events

The DataRepository class supports a number of additional events in a Silverlight application. These are listed in the following table.

Event Name	Description
BoundButtonBeforeAction	Raised just before an action associated with a button is initiated.
BoundButtonAfterAction	Raised just after an action associated with a button has completed.
BoundButtonError	Raised if an error is encountered during the execution of an action associated with a button.

Using the NavigationAssistant Class

mv.NET provides a utility class called NavigationAssistant. This class is used internally by the Silverlight runtime components of mv.NET but it is also available for use by developers. This chapters explains how you can take advantage of the functionality offered by this class

Introduction to the NavigationAssistant Class

When the XAML Wizard is used to create an application navigation design, the code behind generated by the wizard utilizes an instance of the NavigationAssistant class:

VB

```
' Instantiate our NavigationAssistant instance and hook in event handlers
'
NavigationAssistant = New NavigationAssistant(LayoutRoot)
AddHandler NavigationAssistant.MenuOptionSelected, AddressOf MenuOptionSelected

' Make the NavigationAssistant available to the rest of the application via the current
' Application instance
'
NavigationAssistant.AddToApplication(Application.Current)
```

C#

```
// Instantiate our NavigationAssistant instance and hook in event handlers
//
navigationAssistant = new NavigationAssistant(LayoutRoot);
navigationAssistant.MenuOptionSelected += MenuOptionSelected;

// Make the NavigationAssistant available to the rest of the application via the current
Application instance
//
navigationAssistant.AddToApplication(Application.Current);
```

The above code establishes a NavigationAssistant instance, passing in our root visual component on the form – which will typically be a Grid control. Next, an event handler is attached to the MenuOptionSelected event to allow us to respond to the user

selecting menu options from within the application. Finally, we add our NavigationAssistant instance to the Application.Current instance – this allows us to access this NavigationAssistant instance from all areas of our application. The following sections describe why we might want to do this.

On the page in which this code resides, the NavigationAssistant instance is responsible for managing various aspects of the menu display as well as managing the tabbed MDI (Multiple Document Interface) display of forms within the client space of the browser.

Managing the Display of Application Forms

The NavigationAssistant class provides a series of methods which can be used to manage the display of application forms within the client area of the application – these are described below.

OpenForm

This method opens a new form within a new tab in the client area of the application. The arguments to this method are detailed below:

Argument Name	Data Type	Description
FormInstance	Object	The instance of the new form to be displayed.
TabCaption	String	The text to be displayed within the tab ear of the new tab created to host the new form.
ShowMinimizeButton	Boolean	Indicates whether the minimize icon is to be displayed within the new tab ear.
ShowCloseButton	Boolean	Indicates whether the close window icon is to be displayed within the new tab ear.

ShowForm

This (function) method selects an existing form as the currently selected tab within the client area of the application. If the specified form is currently minimized it will be restored back to normal display. The form can be identified by either the text within the relevant tab ear or by a reference to the form instance. The arguments to both overloads of this method are detailed below:

Show form by name

Argument Name	Data Type	Description
FormName	String	The text displayed within the tab ear of the tab hosting the form.

Show form by instance

Argument Name	Data Type	Description
ExistingFormInstance	Object	The instance of the form to show.

If the specified form is not present within the list of forms currently displayed, this method returns a value of "false" as its return value. It is, therefore, designed to be used in conjunction with the OpenForm method to allow you to support scenarios where only a single instance of a given form is to be displayed at any one time.

CloseForm

This method closes a currently active form within the client area of the application. The form can be identified by either the text within the relevant tab ear or by a reference to the form instance. The arguments to both overloads of this method are detailed below:

Close form by name

Argument Name	Data Type	Description
FormName	String	The text displayed within the tab ear of the tab hosting the form.

Close form by instance

Argument Name	Data Type	Description
ExistingFormInstance	Object	The instance of the form to close.

ActiveForms

This method returns a string array containing the name (tab ear text) of all active forms.

FormIsActive

This method allows you to determine whether a certain form is currently active (i.e. contained within a tab). The arguments of this method are detailed below:

Argument Name	Data Type	Description
FormName	String	The text displayed within the tab ear of the tab hosting the form.
FormInstance	Object	The instance of the form if it is currently active.

Using Popup Windows

The NavigationAssistant class provides a popup window mechanism which can be used to display either modal or non-modal popup dialog windows within an application.

In order to display a popup window you need 2 things:

1. A reference to the NavigationAssistant instance created as per described the previous section.
2. An instance of a data form to be displayed in the popup window. This can (although doesn't have to) be a data form created using XAML Wizard.

In order to obtain a reference to the previously created NavigationAssistant instance, you need to use the following code:

```
NavigationAssistant.Instance(Application.Current)
```

Once you have a reference to the NavigationAssistant instance you can use its OpenPopupWindow method to display a popup window. The following lines of code illustrate this usage:

VB

```
Dim contactLookup As New ContactDetails(CType(dgrOrganizations.SelectedItem,
                                                Organization).MainContactID)
NavigationAssistant.Instance(Application.Current).OpenPopUpWindow(True, Me, contactLookup,
                                    True, "Contact Details", True, False, -1, -1, 400, 200)
```

C#

```
ContactDetails contactLookup = new ContactDetails(CType(dgrOrganizations.SelectedItem,
                                                Organization).MainContactID);
NavigationAssistant.Instance(Application.Current).OpenPopUpWindow(true, this, contactLookup,
                                    true, "Contact Details", true, false, -1, -1, 400, 200);
```

The first line of code creates an instance of a ContactDetails form, passing in the item ID of the contact to be displayed in the form. In this case, the ContactDetails form has been created initially using the XAML Wizard.

The second line of code uses the NavigationAssistant instance's OpenPopupWindow method to display the popup window. The arguments supplied to this method are as follows:

Argument Name	Data Type	Description
Modal	Boolean	Indicates whether the popup window is to be displayed in a modal style, i.e. if set to True, the underneath application content will be inaccessible whilst the popup window is active.
ParentForm	Object	The instance of the parent form.

PopUpContent	Object	The content to be displayed in the popup window.
AllowResizing	Boolean	Indicates whether the user can alter the size of the popup window.
WindowCaption	String	The words to appear in the top caption bar of the popup window.
ShowCloseIcon	Boolean	Indicates whether the Close button is to be displayed in the top right corner of the popup window
ShowMaximizeIcon	Boolean	Indicates whether the Minimize button is to be displayed in the top right corner of the popup window
InitialLeftPosition	Int64	The initial left edge position of the popup window (relative to the left edge of the parent window).
InitialTopPosition	Int64	The initial top edge position of the popup window (relative to the top edge of the parent window).
InitialWidth	Int64	The initial width of the popup window.
InitialHeight	Int64	The initial height of the popup window.

The OpenPopupWindow returns a reference to the newly displayed popup window. If you need to get hold of a piece of data from within the popup window when it is closed you can hook into the popup window's "Close" event, as shown below:

VB

```
Dim contactLookup As New ContactDetails(CType(dgrOrganizations.SelectedItem,
                                                Organization).MainContactID)
Dim contactLookupPopup As PopUpWindow = NavigationAssistant.Instance(Application.Current)
                                        .OpenPopUpWindow(True, Me, contactLookup, True,
                                        "Contact Details", True, False, -1, -1, 400, 200)
AddHandler ProductLookupPopup.Closed, AddressOf ContactLookup_Closed

Private Sub ContactLookup_Closed(ByVal Sender As PopUpWindow, ByVal Result As Object)

    If Result IsNot Nothing Then
        ' Result holds the "return" data
    End If

End Sub
```

C#

```
ContactDetails contactLookup = new
             ContactDetails (((Organization)dgrOrganizations.SelectedItem).MainContactID);
PopUpWindow contactLookupPopup =
             NavigationAssistant.Instance(Application.Current).OpenPopUpWindow(true, this,
             contactLookup, true,"Contact Details", true, false, -1, -1, 400, 200);
ProductLookupPopup.Closed += ContactLookup_Closed;
```

```
private void ContactLookup_Closed(PopUpWindow Sender, object Result)
{
    if (Result != null)
        {
            // Result holds the "return" data
        }
}
```

The value of the "Result" argument passed into the "Closed" event is set by your code when you close the popupwindow using the "Close" method of the PopUpWindow class:

VB

```
PopUpWindow.Close(Me, someReturnData)
```

C#

```
PopUpWindow.Close(this, someReturnData);
```

The Sample Silverlight Application

A sample Silverlight application is installed when mv.NET's CIDSetup.exe routine is run. This chapter walks you through the structure of this sample application. We strongly recommend that you look at this sample application as it illustrates many of the topics of interest when developing Silverlight applications in conjunction with mv.NET.

Sample Application Location

The sample Silverlight application is in the following folder:

On modern systems it is in:

```
C:\ProgramData\BlueFinity\mv.NET\Version4.0\Examples\Visual Studio 2010\Silverlight
```

On legacy systems it is in:

```
C:\ Documents and Settings\All Users\Application
Data\BlueFinity\mv.NET\Version4.0\Examples\Visual Studio 2010\Silverlight
```

Pre-requisites for Running the Sample Application

Before you can run the sample Silverlight application you will need to download the sample SOP database. Please refer to the Core Objects developer guide for details on how to do this.

Installing the Sample Application's EMR Definition

If you want to extend the entity model used by the sample application, you will need to download the entity model repository details into a virgin EMR. This is done using mv.NET's Data Transfer utility.

Within the sample application folder is a folder called "EMR Export". This folder holds a Data Transfer export of the relevant EMR file data. You need to use the Data Transfer utility to import all the files in the EMR Export folder into a virgin EMR account. Please refer to the Solution Objects guide for further details on how to create an EMR account,

Once the EMR file content has been imported you will be able to modify its content using the Data Manager.

Sample Application Solution Structure

The sample application consists of 4 projects within Visual Studio. These are:

- **Data Access Browser** – the data access layer used inside the Silverlight environment
- **Data Access Web Service**– the data access layer used inside the web service
- **SilverlightSOPExample** – the application 's interface (runs within the Silverlight environment)
- **SilverlightSOPExample.Web** – The web site hosting the web service used to provide database access to the above project

Normally you wouldn't have the data access projects within the same solution as your end application, however, to aid distribution and ease of understanding this has been done in this case.

XAML Wizard Designs used in the Application

Much of the sample application has been created using the XAML Wizard. Therefore, the application navigation and form designs created as part of this task are included in the sample application folder structure. They can be found in:

`".\VB\XAML Wizard"` or `".\C#\XAML Wizard"`

This allows you to study and modify the designs if desired.