* 1. 

Hands-On Lab

Building Applications in Silverlight 4  
WinForms and Silverlight

**Contents**

[Lab 1: Migrating a Windows Forms Application to Silverlight 3](#_Toc276329485)

[You Will Benefit from this Lab if: 4](#_Toc276329486)

[You Will Learn: 4](#_Toc276329487)

[Business Requirements for the Silverlight application include: 5](#_Toc276329488)

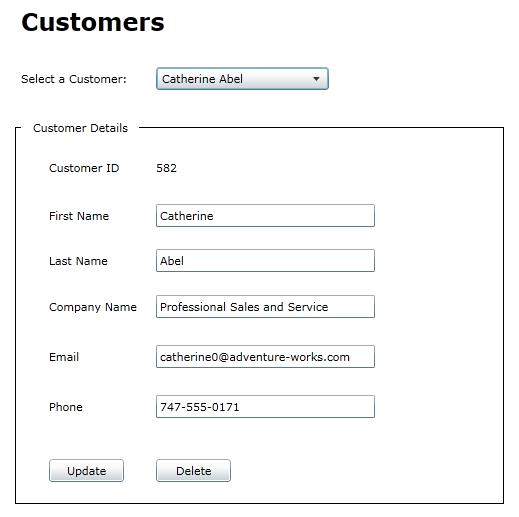
[Exercise 1: Exploring a Windows Forms Application 6](#_Toc276329489)

[Exercise 2: Migrating a Windows Forms Application to Silverlight 10](#_Toc276329490)

[Exercise 3: Calling a WCF Service and Binding Data 20](#_Toc276329491)

[Summary 30](#_Toc276329492)

Lab 1: Migrating a Windows Forms Application to Silverlight

* 1. Silverlight provides a Rich Internet Application (RIA) framework that can be used to build applications that can be deployed through the Web while preserving the rich client-side functionality found in traditional desktop applications created using Windows Forms. This lab is designed to guide Windows Forms developers through the process of migrating applications to Silverlight.
  2. In the lab you'll convert an existing Windows Forms application that consumes data from a Windows Communication Foundation (WCF) service to Silverlight and ensure that existing functionality is preserved. Along the way you'll learn how to create a Silverlight user interface, handle events, make asynchronous calls to services, bind data to controls, plus more.
  3. You'll start by researching existing data access and WCF service projects used by a Windows Forms application. Next, you'll run the Windows Forms application to see the feature set that will be migrated to Silverlight. To accomplish the migration you'll create a new Silverlight project in Visual Studio 2010 and use eXtensible Application Markup Language (XAML) along with managed code (both C# and VB are supported in the lab). The Silverlight application that you'll create is shown next:
  4. 
  5. Figure 1
  6. Customer application UI

You Will Benefit from this Lab if:

* + You are migrating a Windows Forms application to Silverlight
  + You'd like to create a rich-client application that takes advantage of a web-deployment model
  + You need to integrate distributed data into an application

You Will Learn:

* + How to use the Visual Studio 2010 Silverlight Designer
  + XAML and Silverlight control concepts
  + How WCF services can be integrated into Silverlight applications
  + Silverlight data binding techniques
  + How to make asynchronous calls to services
  + How to work with cross-domain services
  + Similarities between Windows Forms and Silverlight applications

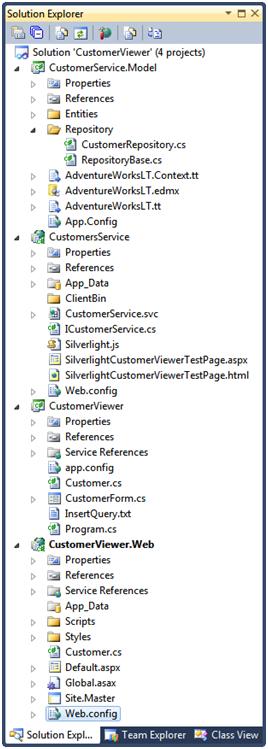
Business Requirements for the Silverlight application include:

* + Create a new Silverlight project and associated ASP.NET Web Application project
  + Re-use existing data access and WCF service code
  + Use XAML to define the user interface and emulate the Windows Forms application
  + Handle user interface events
  + Create a WCF service proxy
  + Call the WCF service using the asynchronous programming model
  + Bind data to controls using Silverlight binding syntax
  + Handle update and delete operations and notify the user about the status of the operation
  1. Estimated Time: 45 minutes

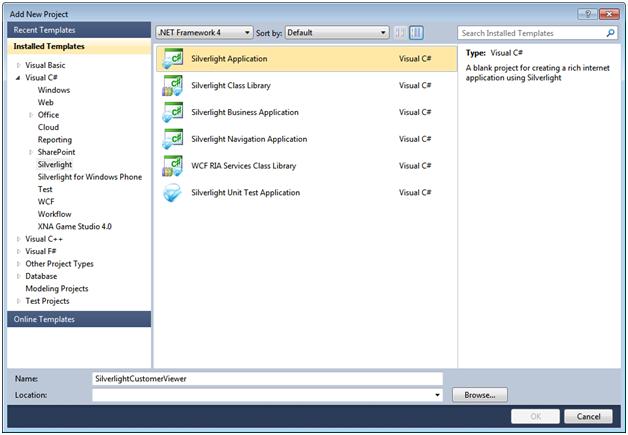
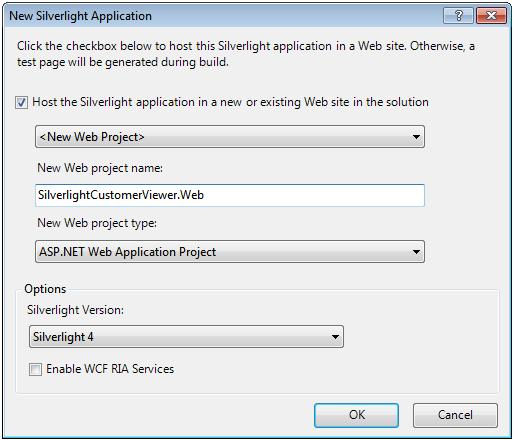
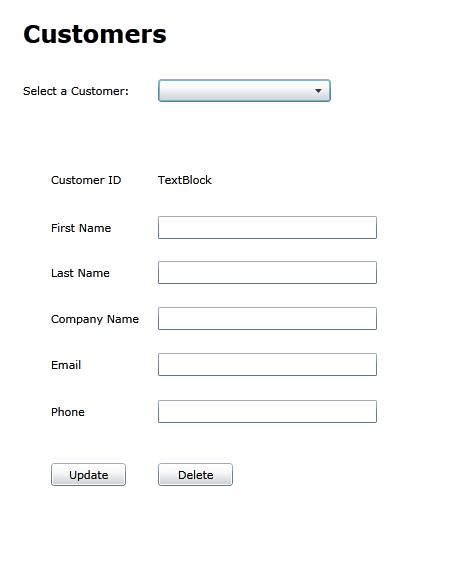
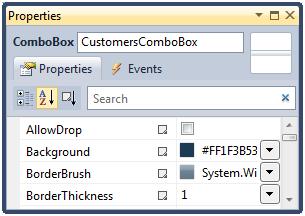
Exercise 1: Exploring a Windows Forms Application

* 1. In this exercise you'll open an existing Visual Studio 2010 solution and walk through code found in a WCF and Windows Forms project. The main goal of the exercise is to get acquainted with an existing Windows Forms application and supporting code to better understand what code can be re-used during the migration to Silverlight. Throughout the exercise you'll view data access code that relies on Entity Framework 4, examine a WCF service contract and run the Windows Forms project to explore the functionality it offers. To get started, follow the steps below.
  2. Open Visual Studio 2010 and select **File 🡪 Open Project/Solution** from the menu.
  3. Open the following Visual Studio solution file:

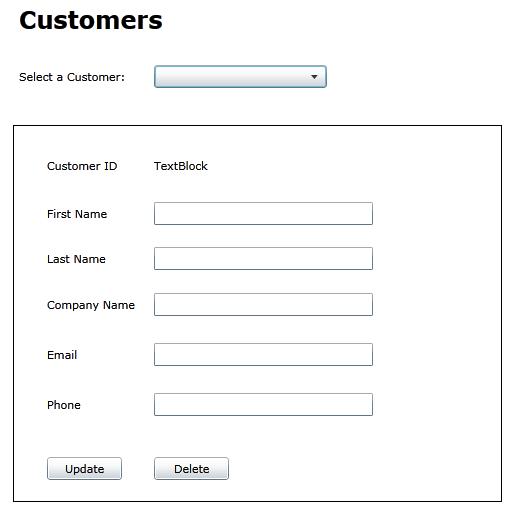
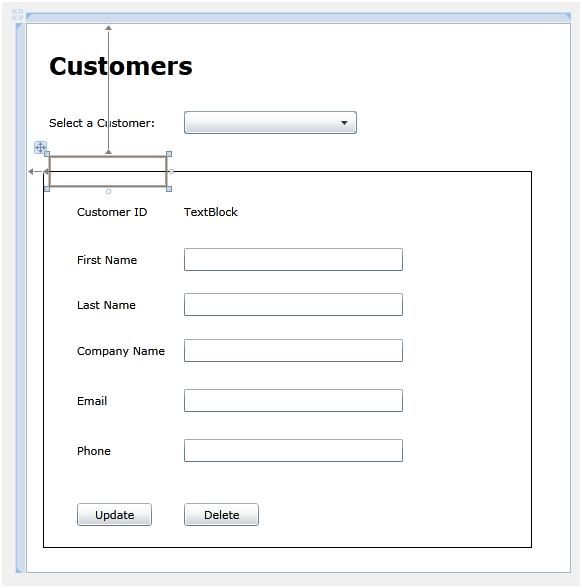
|  |  |
| --- | --- |
| Language | [Lab Files Location] |
| C# | /MigratingToSilverlight/Starting Point/C#/CustomerViewer/CustomerViewer.sln |
| Visual Basic | /MigratingToSilverlight/Starting Point/VB/CustomerViewer/CustomerViewer.sln |

* 1. The following projects are available in the solution:
     + **CustomerService.Model** – Contains entities and data repository classes used to access an AdventureWorks LT database.
     + **CustomersService** – A WCF service application that exposes entities to various applications.
     + **CustomerViewer** – A Windows Forms project that consumes data from a WCF service.
     + **CustomerViewer.Web** – An ASP.NET Web Forms project that uses jQuery to make RESTful calls to a WCF service.
     1. 
     2. Figure 2
     3. Solution explorer view
  2. Right-click on **CustomerService.svc** in the **CustomersService** project and select **View in Browser** from the menu. This will start a local WCF server and show a test page.
  3. Back in Visual Studio, right-click on the **CustomerViewer** project and select **Set as StartUp Project** from the menu.
  4. Run the application by pressing F5. The first time the application runs there will be short delay before data is loaded.
  5. Once data loads, notice that customers appear in the **ComboBox**. Once a customer is selected the details are shown in the form allowing customer data to be updated or deleted.
  6. Back in Visual Studio, right-click on **CustomerForm.cs** or **CustomerForm.vb** (depending upon your language) and select **View Code** from the menu. Take a moment to explore the code and note the following:
     1. A WCF service proxy is used to call a service that supplies customer data
     2. Control data bindings are defined in the **SetBindings** method
     3. Customer data can be updated and deleted through interactions with the WCF service
  7. Locate the **CustomerService.Model** project and double-click the **AdventureWorksLT.edmx** file to see the Entity Framework 4 model that's exposed. The entity model contains several entities including Customer which is used by the Windows Forms application.
     1. **Note:** DataSets are not used in the application since data will exposed through a Web Service and consumed by many different types of clients. The CustomerService.Model project contains strongly-typed objects that work well in a service-oriented environment. Strongly-typed objects also fit in well with Silverlight applications where the DataSet class and related classes such as DataTable aren't supported. By using a Web Service to expose strongly-typed data, many different types of applications can interact with the data regardless of technology or framework.
  8. Open **CustomerRepository** in the **Repository** folder and take a moment to look through the code that interacts with the entity model (you might also want to look at the base class named RepositoryBase). This class is responsible for all communication with Entity Framework and acts as a re-useable repository layer in the application.
  9. Locate the **CustomerService** project and view **ICustomerService** in the editor to see the operations it exposes. The operations are used to load Customer objects and handle update and delete operations. The Windows Forms project currently uses a WCF service proxy object to communicate with the different service operations and the Silverlight project will need to call the same service. Service calls are forwarded to the CustomerRepository class examined earlier.
     1. **Note:** WCF services work well in environments where data must be exposed to different types of clients without requiring a specific technology or framework. The application shown in this lab uses WCF services to promote data re-use, allow different types of clients to consume data, and provide a standards-compliant way to access data.

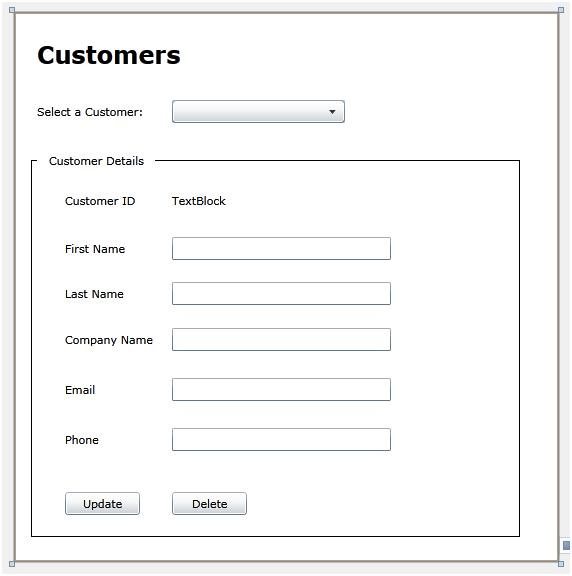
Exercise 2: Migrating a Windows Forms Application to Silverlight

* 1. Now that you've examined the functionality provided by the existing Windows Forms application you'll migrate the application to Silverlight. In this exercise you'll create a new Silverlight project, work with eXtensible Application Markup Language (XAML), create a WCF service proxy to interact with the service and design a user interface that mirrors the existing Windows Forms user interface.
  2. What benefits does XAML offer to an experienced Windows Forms developer? In a nutshell, XAML provides a declarative way to create user interfaces and provides a productive and flexible way to layout controls. XAML allows you define the overall layout of controls used in an application without having to use a programming language such as C# or VB which simplifies development reduces maintenance costs, allows for better re-use of styles, and leads to greater overall productivity. You can also animate, scale, rotate and even skew objects using XAML which opens up many new avenues for presenting data to end users. You can even completely re-design the look and feel of controls using XAML through the use of styles and templates. Finally, data bindings between controls and object properties can be defined declaratively in XAML providing a simple yet robust way to bind data in applications.
  3. Add a new **Silverlight Application** into the solution by right-clicking the **CustomerViewer** solution and selecting **Add 🡪 New Project** from the menu.
  4. From the Installed Templates area on the left of the dialog, pick your desired language (Visual Basic or C#) and select **Silverlight**. Select **Silverlight Application** from the available templates:
     1. 
     2. Figure 3
     3. New project dialog box
  5. Name the project **SilverlightCustomerViewer** and save it within the existing **CustomerViewer** solution folder.
  6. In the dialog window that appears ensure that **<New Web Project>** is selected from the drop-down options and ensure that the project is named **SilverlightCustomerViewer.Web** as shown next. This project will be used to host the Silverlight application in a web page.
     1. 
     2. Figure 4
     3. New SilverLight Application dialog box
  7. Once the project loads you'll see the Visual Studio editor open in split-view mode with a designer on top and a XAML code editor window on the bottom.
  8. Locate the XAML code editor window and change the **UserControl** element's **d:DesignHeight** and **d:DesignWidth** attributes and add **Width** and **Height** attributes as shown next:
     1. XAML
     2. <UserControl x:Class="SilverlightCustomerViewer.MainPage"  
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
            xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
            xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
            mc:Ignorable="d"  
            **d:DesignHeight="545" d:DesignWidth="550" Width="545" Height="550"**>  
            <Grid x:Name="LayoutRoot" Background="White">  
          
            </Grid>  
        </UserControl>
     3. **Note:** The d:DesignHeight and d:DesignWidth attributes control the size of the design surface while in design mode. However, they don't have any effect at runtime. The Height and Width attributes constrain the size of the Silverlight screen at runtime. If you don't supply a Height and Width attribute Silverlight will automatically fill the entire area of its container.
  9. Now that the designer has been resized, drag 9 **TextBlock**controls, 1 **ComboBox** control, 5 **TextBox** controls and 2 **Button** controls from the Toolbox onto the designer and arrange them as shown next.
     1. **Note:** The TextBlock control is analogous to the Label control in Windows Forms. The Silverlight Toolkit (available from http://silverlight.codeplex.com) also provides a Label control that can be used in Silverlight applications.
     2. **Note:** Once you've added a control onto the design surface you can select it and then copy and paste it onto the design surface to add another control of the same type quickly and easily.
     3. 
     4. Figure 5
     5. Silverlight Customer UI
  10. Modify the **Text** property of each **TextBlock** control to match the user interface shown previously (right-click on the control and select **Properties** from the menu).
  11. Modify the **Content** property of each **Button** control to match the user interface shown earlier.
  12. Right-click on the **ComboBox** control, select **Properties** and change the name of the control to a value of **CustomersComboBox** (you can change the name using the text box at the top of the Properties window as shown next):  
      1. 
      2. Figure 6
      3. ComboBox Properties
  13. Change the **DisplayMemberPath** property of the **ComboBox** to a value of **FullName**.
      1. **Note:** DisplayMemberPath is used to define the property that will be displayed as the ComboBox binds to a collection of objects such as Customer objects.
  14. Give the following names to the update and delete buttons in the interface using the Properties window:

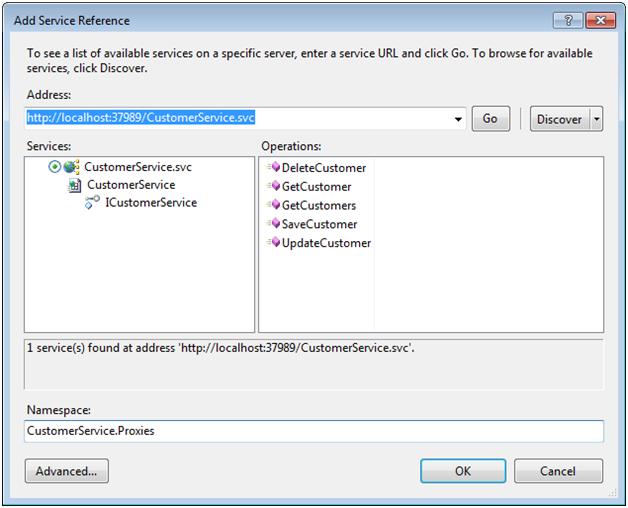
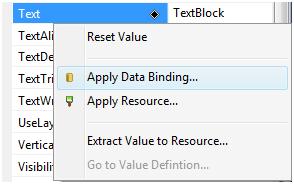
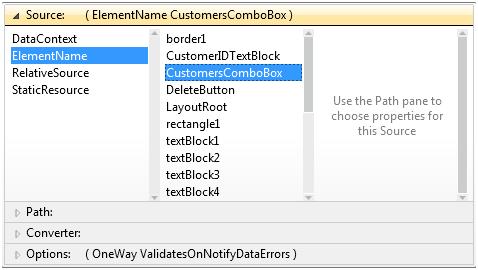
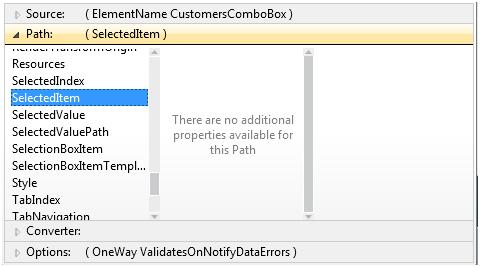
|  |  |
| --- | --- |
| Button Content | Button Name |
| Update | UpdateButton |
| Delete | DeleteButton |

* 1. To simulate an HTML frameset tag or a Windows Forms GroupBox container control, drag a **Rectangle** from the Toolbox and drop it on the designer surface.
  2. Right-click on the **Rectangle** and select **Order 🡪 Send to Back** from the menu.
  3. Resize and arrange the Rectangle so that it encompasses the controls as shown next:
     1. 
     2. Figure 7
     3. Silverlight Application UI
  4. Drag a **Border** control onto the design surface and place it as shown next:
     1. 
     2. Figure 8
     3. Silverlight Application UI
  5. Right-click on the **Border** control in the designer and select **Properties** from the menu.
  6. Change the following properties on the **Border** control:

|  |  |
| --- | --- |
| Property | Value |
| Background | White |
| BorderBrush | White |

* 1. Drag a **TextBlock** control from the Toolbox and drop it **into** the **Border** control (ensure that it's dropped inside of the Border control).
  2. Change the **TextBlock's Text** property to a value of **Customer Details**.
  3. Right-click on the Customer Details **TextBlock** and select **Reset Layout 🡪 Size** from the menu.
  4. The user interface should look like the following once completed:
     1. 
     2. Figure 9
     3. Finished Silverlight Application UI

Exercise 3: Calling a WCF Service and Binding Data

* 1. In this exercise you'll create a WCF service proxy that can be used to call an existing WCF service. You'll also use a clientaccesspolicy.xml file to handle cross-domain issues and bind data to controls.
  2. Right-click on the **SilverlightCustomerViewer** project and select **Add Service Reference** from the menu.
  3. Once the **Add Service Reference** dialog appears click the **Discover** button to locate WCF services within the solution.
  4. Click on the icon to the left of **CustomerService.svc** to expand it. Drill-down until you can see the **ICustomerService** contract. Click the contract name and note that it has several service operations available.
  5. In the **Namespace** box supply a name of **CustomerService.Proxies** as shown next:
     1. 
     2. Figure 10
     3. Add Service Reference dialog box
  6. Click **OK** to create the WCF service proxy.
  7. Add a new **Customer** class into the **SilverlightCustomerViewer** project and change the class's namespace so that it matches with the namespace of the class generated by the WCF proxy:
     1. C#
     2. SilverlightCustomerViewer.CustomerService.Proxies
     3. Visual Basic
     4. CustomerService.Proxies
  8. Add a **FullName** property into the **Customer** class as shown next. Ensure that you mark the class with the **partial** keyword. This property will be displayed by the **ComboBox** control.
     1. C#
     2. public partial class Customer
     3. {
     4. public string FullName
     5. {
     6. get
     7. {
     8. return FirstName + " " + LastName;
     9. }
     10. }
     11. }
     12. Visual Basic
     13. Public Partial Class Customer
     14. Public ReadOnly Property FullName() As String
     15. Get
     16. Return FirstName + " " + LastName
     17. End Get
     18. End Property
     19. End Class
  9. Open the **MainPage.xaml** code-beside file in the editor and import the **proxy** namespace at the top of the code file:
     1. C#
     2. using SilverlightCustomerViewer.CustomerService.Proxies;
     3. Visual Basic
     4. Imports CustomerService.Proxies
  10. Add the following code within the constructor to hook the **Loaded** event to an event handler:
      1. C#
      2. Loaded += MainPage\_Loaded;
      3. Visual Basic
      4. AddHandler Loaded, AddressOf MainPage\_Loaded
  11. Add a **MainPage\_Loaded** method immediately after the constructor with the following code to use the WCF service proxy created earlier and make an asynchronous data request:
      1. C#
      2. void MainPage\_Loaded(object sender, RoutedEventArgs e)
      3. {
      4. var proxy = new CustomerServiceClient();
      5. proxy.GetCustomersCompleted += proxy\_GetCustomersCompleted;
      6. proxy.GetCustomersAsync();
      7. }
      8. Visual Basic
      9. Sub MainPage\_Loaded(sender as Object, e as RoutedEventArgs)
      10. Dim proxy as New CustomerServiceClient()  
           AddHandler proxy.GetCustomersCompleted, \_  
           AddressOf proxy\_GetCustomersCompleted  
           proxy.GetCustomersAsync()
      11. End Sub
  12. Add the following method and associated code to handle the asynchronous callback that will be made when data from the WCF service is returned to the Silverlight application.
      1. **Note:** Once the WCF service proxy returns data it can be accessed through the GetCustomersCompletedEventArgs object's Result property which is typed as an ObservableCollection of Customer. The collection is assigned to the ItemsSource property of the ComboBox which is similar to the DataSource property found on data controls in ASP.NET and Windows Forms.
      2. C#
      3. void proxy\_GetCustomersCompleted(object sender,   
          GetCustomersCompletedEventArgs e)
      4. {
      5. CustomersComboBox.ItemsSource = e.Result;
      6. }
      7. Visual Basic
      8. Sub proxy\_GetCustomersCompleted(sender as Object, \_  
          e as GetCustomersCompletedEventArgs)  
          CustomersComboBox.ItemsSource = e.Result  
         End Sub
  13. Back in **MainPage.xaml**, select the **TextBlock** control immediately to the right of Customer ID and select **Properties** from the menu.
  14. Locate the **Text** property and remove any text from it.
  15. Click on the **Text** property's black triangular icon shown next and select **Apply Data Binding…**from the menu:
      1. 
      2. Figure 11
      3. Apply Data Binding
  16. The data binding properties window will appear and open the **Source** area. Click **ElementName** on the left and **CustomersComboBox** on the right to identify the **ComboBox** as the data binding source as shown next:
      1. 
      2. Figure 12
      3. Setting the Binding Source
  17. Click on the **Path** area (immediately below the Source area of the data binding window) and select **SelectedItem** from the properties:
      1. 
      2. Figure 13
      3. Path
      4. **Note:** After removing the text from the **Text** property you won't see the **TextBlock** control on the designer surface. To get to it within the **Properties** window you can select the **TextBlock** in the XAML code or right-click on the designer and select **Document Outline** from the menu. You can then drill-down into the visual tree to select different controls within the designer and access their properties within the Properties window.
  18. Locate the **TextBlock** control modified within the previous steps in the XAML editor and change the **Text** property value to the following (notice the inclusion of the **CustomerID** property after SelectedItem):
      1. XAML
      2. Text="{Binding ElementName=CustomersComboBox, Path=SelectedItem.**CustomerID**}"
  19. Perform the previous steps to add data bindings to all of the **TextBox** controls in the designer. You'll need to modify the **Text** property of each control within the XAML as in the previous step to specify the appropriate property of the **SelectedItem** to bind to. The properties that each TextBox should bind to are shown next:

|  |  |
| --- | --- |
| TextBox | Property to Bind |
| First Name | FirstName |
| Last Name | LastName |
| Company Name | CompanyName |
| Email | EmailAddress |
| Phone | Phone |

* + 1. **Note:** Once you've defined a binding for one TextBox control's Text property using the data binding window it's often faster to copy and paste it within the XAML to the other TextBox controls and then change the property name defined in the binding.
  1. After adding the proper data bindings to the **TextBox** controls take a moment to look through the XAML and notice that each **TextBox** binding has **Mode=TwoWay** added to it. This allows changes to a **TextBox** control to be propagated back to the bound property automatically. An example of a **TwoWay** binding is show next:
     1. XAML
     2. Text="{Binding ElementName=CustomersComboBox,   
         Path=SelectedItem.FirstName,**Mode=TwoWay**}"
  2. Right-click the **SilverlightCustomerViewer.Web** project and set it as the startup project. Set the html page in the project as the startup page by right-clicking the file and selecting **Set As Start Page**.
  3. Press F5 to compile and run the project and notice that an error occurs once the Silverlight application loads. This is due to a cross-domain call being made from Silverlight to the WCF service. The service uses a different port then the Silverlight host Web project causing a cross-domain exception to be thrown.
  4. To fix the cross-domain issue, rename the existing **clientaccesspolicy.exclude** file in the **CustomersService** project to **clientaccesspolicy.xml**.
  5. Open the **clientaccesspolicy.xml** file in the editor and take a moment to look through the XML. Anytime Silverlight makes a call to a service in a different domain a client access policy file must be in place to successfully talk with the service. This file must be placed at the root of the service application.
  6. Run the application again and notice that data now loads in the **ComboBox** control. Select a customer and notice how the data from it is bound to the appropriate **TextBlock** and **TextBox** controls.
  7. Back in Visual Studio, double-click on both buttons in the designer to create **Click** event handlers.
  8. Add the following code into the **Update** button's click event handler to call the WCF service and pass the updated Customer object:
     1. C#
     2. var proxy = new CustomerServiceClient();
     3. var cust = CustomersComboBox.SelectedItem as Customer;
     4. cust.ChangeTracker.State = ObjectState.Modified;
     5. proxy.SaveCustomerCompleted += (s, args) =>
     6. {
     7. var opStatus = args.Result;
     8. string msg = (opStatus.Status) ? "Customer Updated!" :
     9. "Unable to update Customer: " + opStatus.Message;
     10. MessageBox.Show(msg);
     11. };
     12. proxy.SaveCustomerAsync(cust);
     13. Visual Basic
     14. Dim proxy as New CustomerServiceClient()
     15. Dim cust = CType(CustomersComboBox.SelectedItem, Customer)
     16. cust.ChangeTracker.State = ObjectState.Modified
     17. AddHandler proxy.SaveCustomerCompleted, Sub(s, args)
     18. Dim opStatus = args.Result
     19. Dim msg As String = If(opStatus.Status, "Customer Updated!", \_  
          "Unable to update Customer: " +opStatus.Message)
     20. MessageBox.Show(msg)
     21. End Sub
     22. proxy.SaveCustomerAsync(cust)
  9. Add the following code into the **Delete** button's click event handler:
     1. C#
     2. var proxy = new CustomerServiceClient();
     3. var cust = CustomersComboBox.SelectedItem as Customer;
     4. cust.ChangeTracker.State = ObjectState.Deleted;
     5. proxy.SaveCustomerCompleted += (s, args) =>
     6. {
     7. OperationStatus opStatus = args.Result;
     8. if (opStatus.Status)
     9. {
     10. ((ObservableCollection<Customer>)CustomersComboBox.ItemsSource).Remove(cust);
     11. MessageBox.Show("Customer deleted!");
     12. }
     13. else
     14. {
     15. MessageBox.Show("Unable to delete Customer: " + opStatus.Message);
     16. }
     17. };
     18. proxy.SaveCustomerAsync(cust);
     19. Visual Basic
     20. Dim proxy as New CustomerServiceClient()
     21. Dim cust = CType(CustomersComboBox.SelectedItem, Customer)
     22. cust.ChangeTracker.State = ObjectState.Deleted
     23. AddHandler proxy.SaveCustomerCompleted, Sub(s, args)
     24. Dim opStatus As OperationStatus = args.Result
     25. If opStatus.Status Then
     26. CType(CustomersComboBox.ItemsSource, \_  
          ObservableCollection(Of Customer)).Remove(cust)
     27. MessageBox.Show("Customer deleted!")
     28. Else
     29. MessageBox.Show("Unable to delete Customer: " + opStatus.Message)
     30. End If
     31. End Sub
     32. proxy.SaveCustomerAsync(cust)
  10. Run the application and test the update and delete functionality.

Summary

* 1. In this exercise you examined an existing Windows Forms application and supporting data access and service layers. You then migrated the existing functionality in the Windows Forms application to Silverlight and satisfied the following requirements:
  + Create a new Silverlight project and associated ASP.NET Web Application project
  + Re-use existing data access and WCF service code
  + Use XAML to define the user interface and emulate the Windows Forms application
  + Handle user interface events
  + Create a WCF service proxy
  + Call the WCF service using the asynchronous programming model
  + Bind data to controls using Silverlight binding syntax
  + Handle update and delete operations and notify the user about the status of the operation
  1. Although the application created in this lab demonstrates how XAML and managed code can be used, other labs will provide additional details about application design practices that can be followed such as the Model-View-ViewModel (MVVM) pattern that you can use to build Silverlight applications.