* 1. 

Hands-On Lab

Building Applications in Silverlight 4

Using WCF RIA Services

**Contents**

[Lab 08: Using WCF RIA Services 3](#_Toc276336290)

[Exercise 1: Creating a WCF RIA Services Domain Service 6](#_Toc276336291)

[Exercise 2: Creating the User Interface 11](#_Toc276336292)

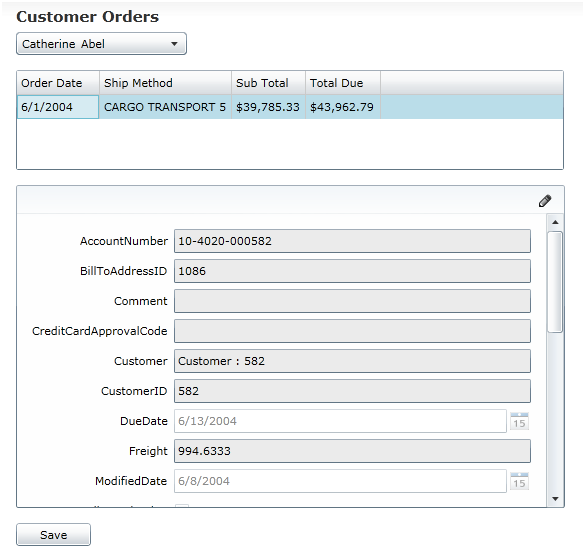
[Exercise 3: Calling a Domain Service from a Silverlight Client 15](#_Toc276336293)

[Exercise 4: Using WCF RIA Services MetaData Classes 19](#_Toc276336294)

[Exercise 5 (Optional): Walkthrough - Creating a Custom Domain Service 22](#_Toc276336295)

[Summary 23](#_Toc276336296)

Lab 08: Using WCF RIA Services

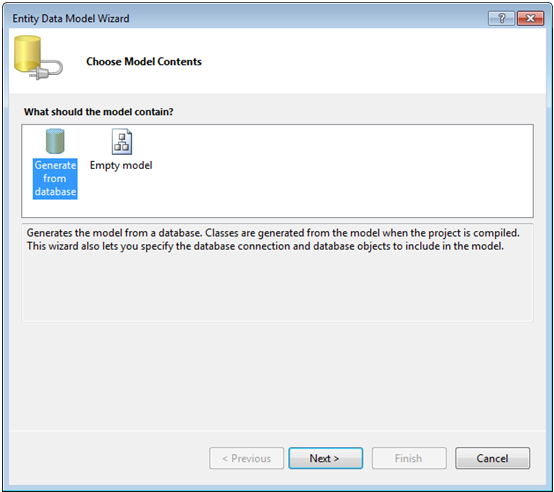
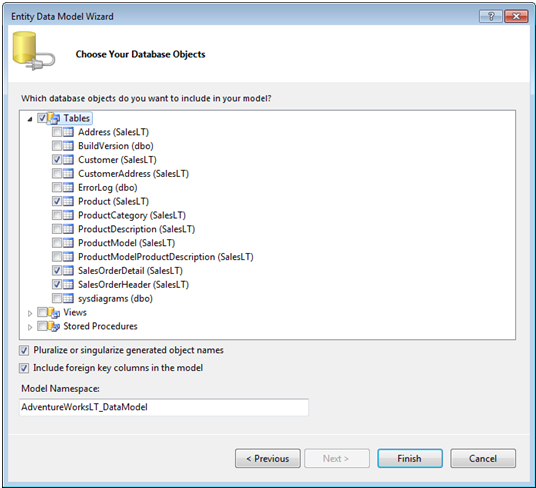
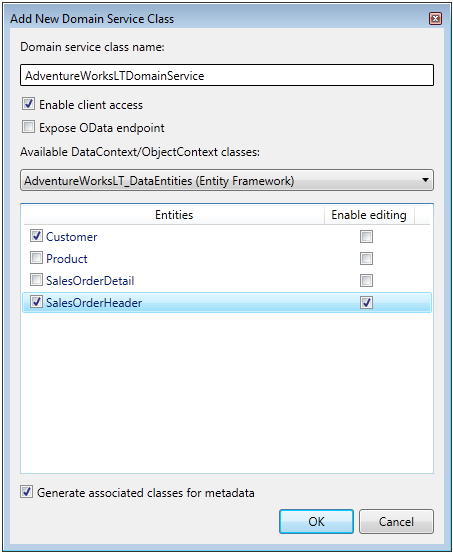
* 1. Silverlight provides a flexible framework for working with distributed data retrieved from different types of services but does require developers to understand asynchronous programming and handle callback operations. Applications that allow users to update, insert and delete data also require developers to perform their own client-side object tracking and to write client and server-side code to validate data. While each of these tasks can be addressed with different coding techniques, using a framework that simplifies these types of tasks can lead to greater overall productivity and fewer bugs.
  2. WCF RIA Services provides a distributed data exchange framework that builds upon existing functionality in Windows Communication Foundation (WCF) to allow data to be exchanged between a Silverlight client and a server. It provides developers with a simplified way to work with the asynchronous programming model, track object state within Silverlight applications, share validation code between client and server projects, build client-side proxy objects, plus more.
  3. In this lab you'll create a WCF RIA Services domain service class and call it from a Silverlight application. You'll also learn how data annotations can be applied to metadata classes to validate object properties and share validation logic across client and server. A bonus exercise is also included to show how WCF RIA Services can be used with custom patterns such as the repository pattern. The application that you'll work with in the lab exercises is shown next:
     1. 
     2. Figure 1
     3. Customer Orders Application
  4. **You Will Benefit from this Lab if:**
  + You're interested in building Silverlight applications that take advantage of WCF RIA Services
  + You're interested in learning how to create a domain service and call it from a Silverlight client
  + You want to share validation logic between a client and a server
  1. **You Will Learn:**
  + How to create an Entity Framework 4 model
  + How to create a WCF RIA Services domain service using the built-in Visual Studio 2010 wizard
  + How to customize domain service methods
  + How to call a domain service from a Silverlight client
  + How to share validation logic between the client and server
  1. **Business Requirements for the Silverlight application include:**
  + Create a new Silverlight Business Application
  + Create a new Entity Framework 4 Model
  + Create a WCF RIA Services domain service class
  + Customize domain service class methods
  + Build a user interface
  + Add code to call the domain service using the generated domain context class
  + Add code to send updates from the Silverlight client to the domain service
  + Create a custom data validation class
  + Add data validation attributes to validate data and share the validation logic between the service and Silverlight client projects

**Estimated Time: 45 minutes**

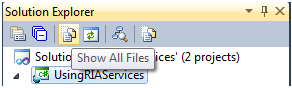
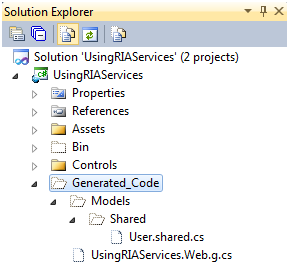
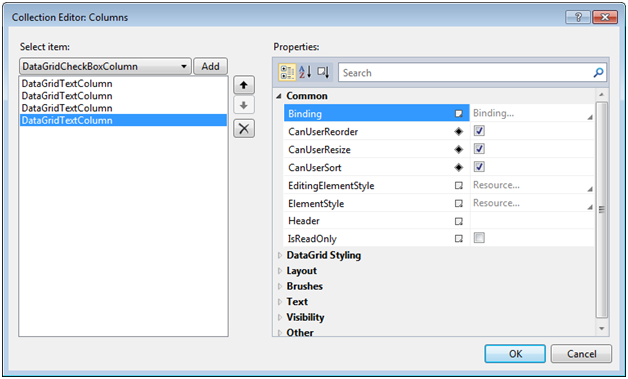
Exercise 1: Creating a WCF RIA Services Domain Service

* 1. In this exercise you'll create a new Silverlight Business Application that has WCF RIA Services features built-in. You'll then create an Entity Framework 4 model in the web project and create a domain service based upon the model. Once the domain service is created you'll customize methods to return more constrained results using LINQ.
  2. Create a new **Silverlight Business Application** in Visual Studio 2010 named **UsingRIAServices**.
     1. **Note:** The Silverlight Business Application template builds upon functionality available in the Silverlight Navigation Application template and also adds WCF RIA Service specific functionality.
  3. Right-click on the **UsingRIAServices.Web** project's **App\_Data** folder and select **Add 🡪 Existing Item** from the menu. Add the following file:

RIAServices/Starting Point/AdventureWorksLT\_Data.mdf

* 1. Add a new **ADO.NET Entity Data Model** into the **WCFRIAServices.Web/Model** folder named **AdventureWorksLT.edmx**. Add it within the **Models** folder.
     1. **Note:** The ADO.NET Entity Data Model template can be found in the Data section. Alternatively, you can use Visual Studio 2010's *Search Installed Templates* feature in the upper-right corner of the dialog window to search for the template as well.
  2. Select **Generate from database** from the options and click the **Next** button.
     1. 
     2. Figure 2
     3. Entity Data Model Wizard
  3. Select **AdventureWorksLT\_Data.mdf** from the drop-down box and click **Next.**
  4. Expand the **Tables** node and select the **Customer**, **Product**, **SalesOrderDetail,** and **SalesOrderHeader** tables as shown next and then click **Finish.**
     1. 
     2. Figure 3
     3. Choose Your Database Objects Dialog
  5. Build the solution
  6. Add a new **Domain Service** **Class** into the **UsingRIAServices.Web/Services** folder and name it **AdventureWorksLTDomainService.cs**.
     1. **Note:** The Domain Service Class template is located in the Web templates section that's available when adding new items into a project.
  7. Once the **Add New Domain Service Class** wizard appears check the checkbox next to the **Customer** and **SalesOrderHeader** entities and enable editing for the **SalesOrderHeader** entity as shown next:
     1. 
     2. Figure 4
     3. Add New Domain Service Class Dialog
     4. **Note:** Ensure that you leave the *Generate associated classes for metadata* checkbox selected. You'll explore this feature later in the lab and use it to add data validation rules into the solution. Notice that the Product and SalesOrderDetail entities aren't checked. Since they won't be used in this application there's no reason to expose them through a WCF RIA Services domain service. You should only expose objects that your application explicitly needs.
  8. Take a moment to examine the **AdventureWorksLTDomainService** class that is created. Notice that it is decorated with the **EnableClientAccess** attribute. This attribute allows Silverlight clients to call it and has several methods added into it to query the entities selected earlier with the wizard.
  9. Locate **GetCustomers** in the domain service and change the code to the following to constrain the query to select customers that have orders and sort customers by last name.
     1. C#
     2. return from c in ObjectContext.Customers  
               join o in ObjectContext.SalesOrderHeaders
     3. on c.CustomerID equals o.CustomerID
     4. orderby c.LastName
     5. select c;
     6. Visual Basic
     7. Return From c In ObjectContext.Customers \_
     8. Join o In ObjectContext.SalesOrderHeaders \_
     9. On c.CustomerID Equals o.CustomerID \_
     10. Order By c.LastName \_
     11. Select c
  10. Rename the **GetSalesOrderHeaders** method to **GetOrdersByCustomerID** and change the method signature to accept a single **Integer** parameter named **customerID**.
  11. Replace the body of the **GetOrdersByCustomerID** method with the following code:
      1. C#
      2. return ObjectContext.SalesOrderHeaders  
          .where(so => so.CustomerID == customerID);
      3. Visual Basic
      4. Return ObjectContext.SalesOrderHeaders.Where(Function(so) \_
      5. so.CustomerID = customerID)
  12. Build the solution before continuing.

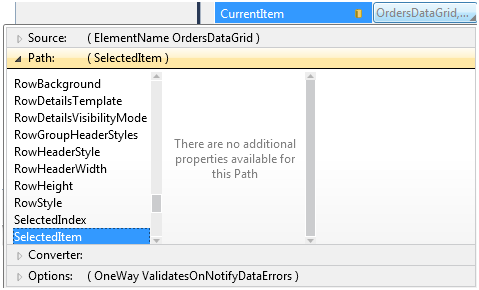
Exercise 2: Creating the User Interface

* 1. In this exercise you'll add controls to a Silverlight user interface and add code to call the domain service created in the previous exercise. Throughout the exercise you'll see how data from a domain service can be accessed and bound to controls using a data context object and see how changes to objects are automatically tracked. You'll also send any changes made in the Silverlight client back to the domain service so that data is updated in the database properly.
     1. **Note:** This exercise uses the DataForm control available in the Silverlight Toolkit. If you don't currently have the Silverlight Toolkit installed, download and install it from http://silverlight.codeplex.com before continuing.
  2. Highlight the **UsingRIAServices** project and click the **Show All Files** icon immediately above the solution name in the Solution Explorer.
     1. 
     2. Figure 5
     3. Show All Files
  3. Notice that a **Generated\_Code** folder appears that contains the code generated by WCF RIA Services.
     1. 
     2. Figure 6
     3. Generated Code
  4. Open the **UsingRIAServices.Web.g** file and take a moment to explore the classes that are in it. Locate the class named **AdventureWorksLTDomainContext** in the file. This class acts as the "proxy" to communicate from the Silverlight client to the domain service.
  5. Open **Home.xaml** (located in the **Views** folder of the **UsingRIAServices** project) in the Visual Studio 2010 designer window and perform the following steps. The layout of the controls on the user interface is shown at the beginning of this lab.
     1. Note: The user interface that you'll create in this task is the same as the one created in the WCF Services lab. If you've already completed that lab, feel free to copy the XAML from this lab's Completed folder and paste it into Home.xaml. You can then skip to Exercise 3.
     2. Change the **TextBlock's Text** property from **Home** to **Customer Orders** (you can do this through the Properties window by resetting the existing Text property value or by typing it directly in the XAML)
     3. Delete the **TextBlock** with a value of **Home page content**
     4. Add a **ComboBox** control under the **TextBlock** and give it a name of **CustomersComboBox**
     5. Drag a **DataGrid** control from the ToolBox and place it under the **ComboBox**. Give it a name of **OrdersDataGrid**
     6. Ensure that the **DataGrid** control's **AutoGenerateColumns** property is set to **False**
     7. Set the **DataGrid** control's **IsReadOnly** property to **True**
  6. Select the **DataGrid** in the designer and locate its **Columns** property in the Properties window. Click the ellipsis button to the right of the **Columns** property to open up the collection editor dialog.
  7. From the dialog window's **Select item** drop-down list select **DataGridTextColumn** and click **Add**. Add a total of 4 columns as shown next:
     1. 
     2. Figure 7
     3. DataGrid Collection Editor
  8. Select each **DataGridTextColumn** and change its **Header** property in the Properties section of the dialog window to one of the following values (Order Date would be assigned to the first column and so on):

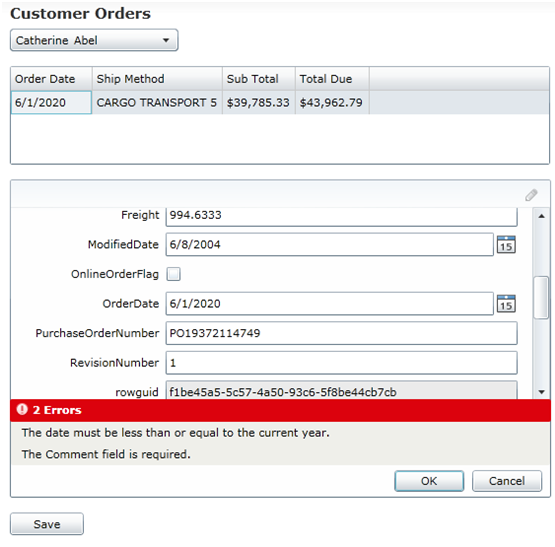
Order Date  
Ship Method  
Sub Total  
Total Due

* 1. Switch to the XAML editor and locate all of the **DataGridTextColumns** elements that have been added.
  2. Add the appropriate **Binding** attribute shown below to each **DataGridTextColumn** based upon the column's header text:
     + 1. Binding="{Binding OrderDate,StringFormat=d}"
       2. Binding="{Binding ShipMethod}"
       3. Binding="{Binding SubTotal,StringFormat=C}"
       4. Binding="{Binding TotalDue,StringFormat=C}"
  3. Add the following template into the **ComboBox** control using the XAML editor:
     1. XAML
     2. <ComboBox.ItemTemplate>
     3. <DataTemplate>
     4. <StackPanel Orientation="Horizontal">
     5. <TextBlock Text="{Binding FirstName}" />
     6. <TextBlock Text="{Binding LastName}" Margin="5,0,0,0" />
     7. </StackPanel
     8. </DataTemplate>
     9. </ComboBox.ItemTemplate>

Exercise 3: Calling a Domain Service from a Silverlight Client

* 1. In this exercise you'll add controls to a Silverlight user interface and add code to call the domain service created in the previous exercise. Throughout the exercise you'll see how data from a domain service can be accessed and bound to controls using a data context object and see how changes to objects are automatically tracked. You'll also send any changes made in the Silverlight client back to the domain service so that data is updated in the database properly.
  2. Handle the **ComboBox** control's **SelectionChanged** event by double-clicking on the control in the designer.
  3. Add the following code above the **Home** class's constructor to create an instance of the **AdventureWorksLTDomainContext** class that will be used to call the domain service. You'll need to resolve the appropriate namespace.
     1. C#
     2. AdventureWorksLTDomainContext \_Context =
     3. new AdventureWorksLTDomainContext();
     4. Visual Basic
     5. Dim \_Context as New AdventureWorksLTDomainContext()
  4. Locate the **OnNavigatedTo** event handler and add the following code:
     1. C#
     2. CustomersComboBox.ItemsSource = \_Context.Customers;
     3. OrdersDataGrid.ItemsSource = \_Context.SalesOrderHeaders;
     4. \_Context.Load(\_Context.GetCustomersQuery());
     5. Visual Basic
     6. CustomersComboBox.ItemsSource = \_Context.Customers
     7. OrdersDataGrid.ItemsSource = \_Context.SalesOrderHeaders
     8. \_Context.Load(\_Context.GetCustomersQuery())
     9. **Note:** This code binds Customer objects to the ComboBox and SalesOrderHeader objects to the DataGrid. It then then calls the domain service's GetCustomers method to fill the Customers collection with data.
  5. Add code in the **SelectionChanged** event handler to perform the following steps:
     1. Get the selected **CustomerID** from the **ComboBox**
     2. Clear any existing objects in the **SalesOrderHeaders** collection
     3. Call the domain service's **GetOrdersByCustomerID** method using the \_**Context** variable.
     4. C#
     5. int custID = ((Customer)CustomersComboBox.SelectedItem).CustomerID;
     6. if (\_Context.EntityContainer != null)  
         \_Context.EntityContainer.GetEntitySet<SalesOrderHeader>().Clear();
     7. \_Context.Load(\_Context.GetOrdersByCustomerIDQuery(custID));
     8. Visual Basic
     9. Dim custID As Integer = (CType(CustomersComboBox.SelectedItem, \_  
        Customer)).CustomerID
     10. If \_Context.EntityContainer IsNot Nothing Then
     11. \_Context.EntityContainer.GetEntitySet(Of SalesOrderHeader)().Clear()
     12. End If
     13. \_Context.Load(\_Context.GetOrdersByCustomerIDQuery(custID))
  6. Run the project and test the Silverlight application in the browser. As a customer is selected all of their orders should show in the **DataGrid** control.
  7. Switch back to **Home.xaml** and drag a **DataForm** control from the ToolBox and place it under the **DataGrid**.
  8. Add the following attributes on the **DataForm** tag by editing the XAML or by using the Properties window.
     1. XAML
     2. CurrentItem="{Binding Path=SelectedItem,ElementName=OrdersDataGrid}"  
        AutoEdit="False"   
        AutoCommit="false"
     3. **Note:** An example of using the Properties window to modify the CurrentItem property visually is shown next. Binding a XAML element such as CurrentItem to another XAML element's property is referred to as *element to element* binding.
     4. 
     5. Figure 8
     6. Current Item Property
     7. **Note:** In this exercise the DataForm will be used to show all properties of the SalesOrderHeader object. In a real-life application you'd want to constrain the data shown by the DataForm control.
  9. Add a **Button** below the **DataForm** and assign a value of **Save** to its **Content** property.
  10. Double-click the **Button** control in the designer to handle the **Click** event.
  11. Within the **Click** event handler add the following code to send any changes made back to WCF RIA Services:
      1. C#
      2. if (\_Context.HasChanges)
      3. {
      4. \_Context.SubmitChanges(so =>
      5. {
      6. string errorMsg = (so.HasError) ? "failed" : "succeeded";
      7. MessageBox.Show("Update " + errorMsg);
      8. }, null);
      9. }
      10. else
      11. {
      12. MessageBox.Show("No changes have been made!");
      13. }
      14. Visual Basic
      15. If \_Context.HasChanges Then
      16. \_Context.SubmitChanges(Sub(so)
      17. Dim errorMsg As String = If(so.HasError, "failed", \_  
           "succeeded")
      18. MessageBox.Show("Update " + errorMsg)
      19. End Sub, Nothing)
      20. Else
      21. MessageBox.Show("No changes have been made!")
      22. End If
  12. Run the project and perform the following tasks in the Silverlight application:
      1. Select a customer and then click on an order in the **DataGrid**
      2. Select the edit icon in the **DataForm** (click the pencil icon in the upper-right hand corner of the control) to switch to edit mode
      3. Change the value for the **Comment** field in the **DataForm** to any value you'd like and click the **OK** button
      4. Click the **Save** button to commit the changes to the database using WCF RIA Services. You should see a success message.
      5. Refresh the browser and navigate to the same customer order. Ensure that the comment change you made was successful.

Exercise 4: Using WCF RIA Services MetaData Classes

* 1. WCF RIA Services provides a simple way to add data validation logic in one place and share the logic between server and Silverlight client projects. In this exercise you'll create a custom data validation class and apply it to a metadata class generated by the WCF RIA Services wizard used earlier in the lab. WCF RIA Services supports several different types of validation attributes including RegularExpression, Range, Required, StringLength and CustomValidation.
  2. Add a new class into the **UsingRIAServices.Web/Services** folder named **AdventureWorksLTDomainService.shared.cs** or **AdventureWorksLTDomainService.shared.vb** depending upon your language.
  3. Replace the default class definition with the following:
     1. C#
     2. public static class DateValidator
     3. {  
        }
     4. Visual Basic
     5. Public NotInheritable Class DateValidator
     6. End Class
     7. **Note:** Classes with ".shared" in the name are automatically shared between the domain service project and the Silverlight project. Once compiled, the DateValidator class will be copied to the Generated\_Code folder that you looked at earlier in the Silverlight project.
  4. Add the following code within the **DateValidator** class to ensure that any date entered has a year less than or equal to the current year (resolve any missing namespaces as needed):
     1. C#
     2. public static ValidationResult ValidateDate(DateTime dt,   
         ValidationContext context)
     3. {
     4. if (dt.Year <= DateTime.Now.Year)
     5. {
     6. return ValidationResult.Success;
     7. }
     8. else
     9. {
     10. return new ValidationResult("The date must be less than " +
     11. "or equal to the current year.");
     12. }
     13. }
     14. Visual Basic
     15. Public Shared Function ValidateDate(ByVal dt As Date, ByVal context As \_  
          ValidationContext) As ValidationResult
     16. If dt.Year <= Date.Now.Year Then
     17. Return ValidationResult.Success
     18. Else
     19. Return New ValidationResult("The date must be less than " + \_  
          "or equal to the current year.")
     20. End If
     21. End Function
     22. **Note:** The ValidationContext and ValidationResult classes reside in the System.ComponentModel.DataAnnotations namespace.
  5. Open the **AdventureWorksLTDomainService.metadata** class in the code editor.
     1. **Note:** This class was automatically added as a result of leaving the *Generate associated classes for metadata* checkbox checked in the Domain Service Class wizard you ran earlier.
  6. Add the following attribute above the **OrderDate** field in the **SalesOrderHeaderMetadata** class to link the **DateValidator** class to the **OrderDate** property for validation purposes:
     1. C#
     2. [CustomValidation(typeof(UsingRIAServices.Web.Services.DateValidator),   
         "ValidateDate")]
     3. Visual Basic
     4. <CustomValidation(GetType(DateValidator), "ValidateDate")>
  7. Add the **Required** attribute above the **Comment** property in the **SalesOrderHeaderMetaData** class to require that a value is entered for a comment.
     1. C#
     2. [Required]
     3. Visual Basic
     4. <Required()>
  8. Run the application, select a customer order and change the **OrderDate** year to a future year such as 2020. Click OK and you'll see errors displayed as shown next:
     1. 
     2. Figure 9
     3. Order Date
     4. **Note:** The error message displayed can be customized by adding the ErrorMessage property to any of the validation attributes.
  9. Fix the data validation errors and save the changes.

Exercise 5 (Optional): Walkthrough - Creating a Custom Domain Service

* 1. In the previous exercises you created and worked with a domain service that only supported Entity Framework 4. While WCF RIA Services supports creating domain services based upon Entity Framework or LINQ to SQL out of the box, you can create custom domain services as well. Going this route allows other ORM frameworks such as nHibernate or PLINQO as well standard ADO.NET code to be used with WCF RIA Services. This exercise will walk through an existing domain service that is ORM neutral allowing for additional flexibility and code abstraction.
  2. Open the following solution in Visual Studio 2010 based upon your chosen language:
     1. C#
     2. RIAServices/Completed/Bonus/C#/UsingRIAServices.sln
     3. Visual Basic
     4. RIAServices/Completed/Bonus/VB/UsingRIAServices.sln
  3. Expand the **UsingRIAServices.Web/Repository** folder and open the **CustomerRepository** class file. Notice that it implements an **ICustomerRepository** interface and provides a single member named **GetCustomers**. The LINQ code within GetCustomers is the same code you added to the domain service in a previous exercise. It's now been moved outside of the domain service to allow for better re-use.
  4. Open the **SalesOrderHeaderRepository** class file and take a moment to look at the members it contains to retrieve and update sales order data.
  5. Expand the **UsingRIAServices.Web/Services** folder and open the **AdventureWorksLTDomainService** class. Note the following features of the class:
     1. The domain service class derives from DomainService as opposed to an Entity Framework or LINQ to SQL base class.
     2. The class creates instances of the repository classes shown earlier in this exercise. These objects could dynamically be injected into the domain service when more loosely coupled code is required.
     3. The domain service operations forward calls to the respective repository classes.
     4. The domain service class does not reference any specific data access framework. This allows developers to use any data access framework they like.
  6. Open the **AdventureWorksLTDomainService.metadata** class file. Locate the **Key** attribute added into the **CustomerMetadata** and **SalesOrderHeaderMetadata** classes. The domain service isn't tied directly to a data access technology so you must specify the key for each object in order for WCF RIA Services to work properly. Because Entity Framework is used in this example the metadata classes provide a good location to apply the Key attribute.

Summary

* 1. In this lab you created an Entity Framework 4 model and exposed specific classes from it to a Silverlight client using a WCF RIA Services domain service. You also called the service using a domain context and added custom data validation logic into the solution. The specific tasks completed are listed next:
  + Create a new Silverlight Business Application
  + Create a new Entity Framework 4 Model
  + Create a WCF RIA Services domain service class
  + Customize domain service class methods
  + Build a user interface
  + Add code to call the domain service using the generated domain context class
  + Add code to send updates from the Silverlight client to the domain service
  + Create a custom data validation class
  + Add data validation attributes to validate data and share the validation logic between the service and Silverlight client projects