

Silverlight 2 Beta 2 Tutorial

Filling a DataGrid using Sql, Linq and WCF

By Jesse Liberty

SQL Data

Many readers of the [second tutorial](#) have written asking how they can use DataBinding to bind to data in their SQL Database. This tutorial will walk through accessing SQL Data by creating a Web Service and then using LINQ to create a data source you can bind to. The control we will bind to will be the **DataGrid**. The DataGrid deserves and will receive its own (upcoming) tutorial as it has quite a lot to offer.

New Skills

The application we'll build will combine a number of different new skills:

- Connecting to a WCF Web Service
- Using LINQ to query and retrieve data that our Silverlight Application can use
- Using the DataGrid control to display data

From a strictly Silverlight perspective, building the Web Service and using LINQ are related skills but beyond the scope of these tutorials. That said, we'll examine how they intersect with Silverlight.

Getting Started

To begin, create a project named SQLData, but be sure to choose **Web Application Project** as we want both a Silverlight project and also a Server project in which we can create a WCF Web Service (to connect to the database).

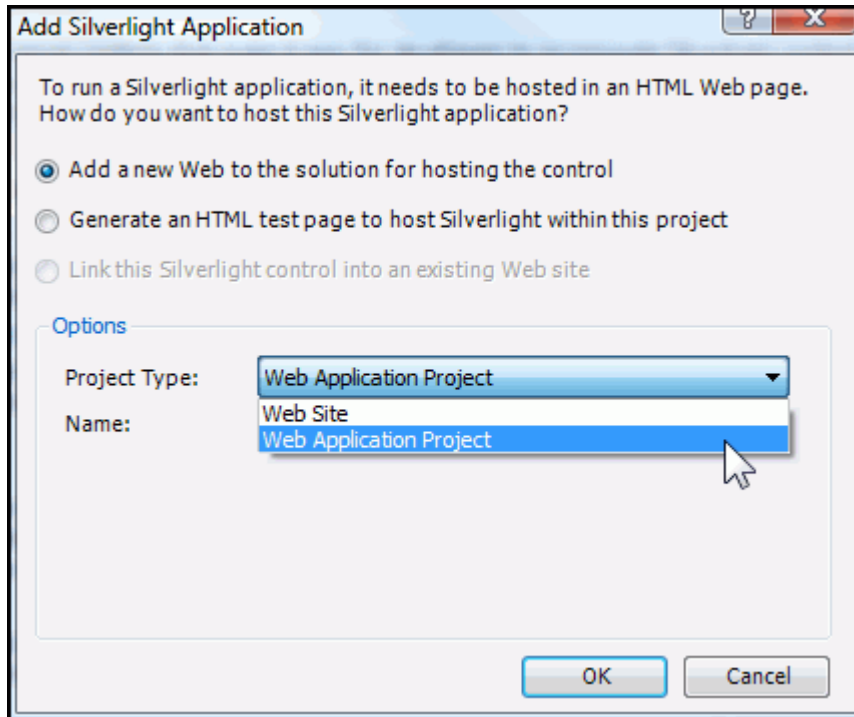


Figure 3-1. Create a Web Application Project

Examining the Two Projects

Visual Studio 2008 will create two projects under one solution.

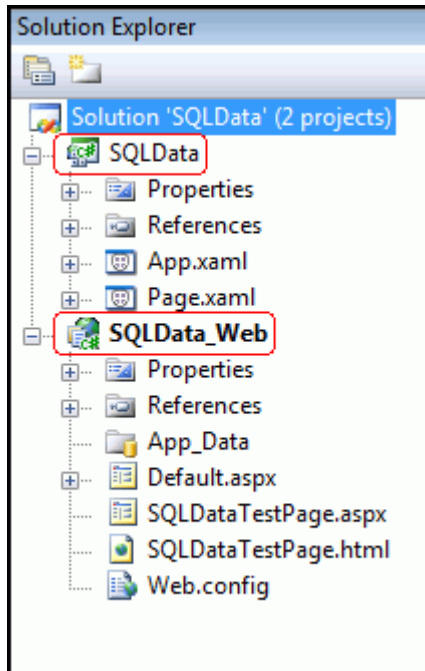


Figure 3-2. Two projects in one solution

The solution and first project are named SQLData. That first project is the Silverlight application and has the same files that you've seen in previous tutorials.

The second project, SQLData_Web is created for you as a test environment for the Silverlight project and it has three potential entry points,

- Default.aspx
- SQLDataTestPage.aspx
- SQLDataTestPage.html

SQLDataTestPage.aspx is specifically designed to test the Silverlight controls and quick examination shows that it includes an AJAX ScriptManager and an ASP:Silverlight control whose source is the .xap file (pronounced zap file) that will be produced by the Silverlight project,

```
<form id="form1" runat="server" style="height:100%;">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div style="height:100%;">
        <asp:Silverlight ID="Xaml1" runat="server" Source="~/ClientBin/SQLData.xap"
            Version="2.0" Width="100%" Height="100%" />
    </div>
</form>
```

Linq To Sql

LINQ is a very powerful addition to both VB 9 and C# 3 and is likely to be a central technique for data retrieval for Silverlight and other .NET technology going forward.

A good way to get started with LINQ is with ScottGu's excellent tutorial available at start is with [ScottGu's tutorial](http://tinyurl.com/28q63z) available at <http://tinyurl.com/28q63z>. There are numerous books on LINQ as well.

In this tutorial we'll be writing a simple LINQ query that I'll parse for you.

To begin right click on the server project and choosing Add, and then choose the LinqToSql Classes template. Notice that the explanation below the window says "LINQ to SQL classes mapped to relational objects."

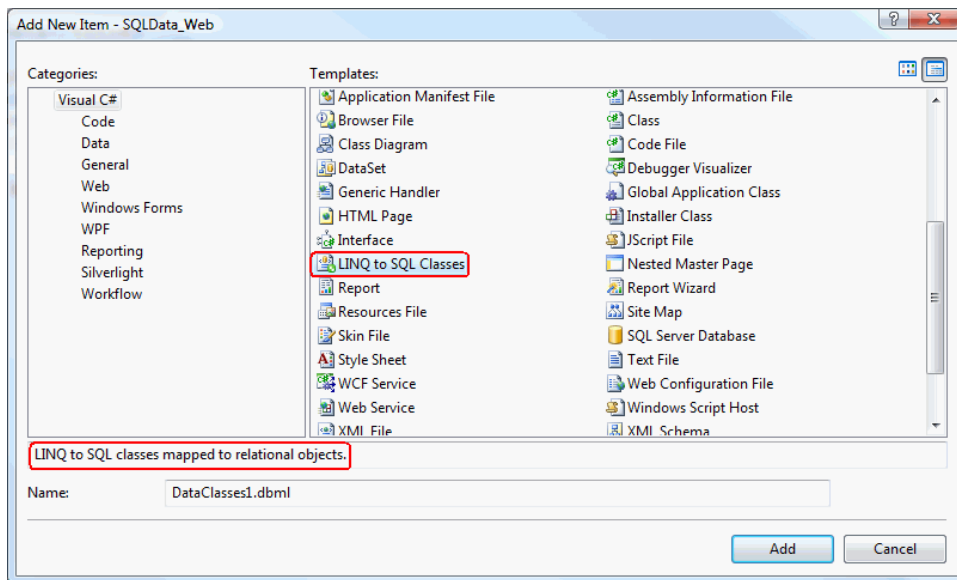


Figure 3-3. Adding LinqToSql Template

When the Object Relational Designer window opens, open the Server Explorer and navigate to the AdventureWorks database (installed with SQL Server or available from Microsoft.com). Expand to reveal the tables and drag the Customer table onto the DataClasses1.dbml Designer workspace,

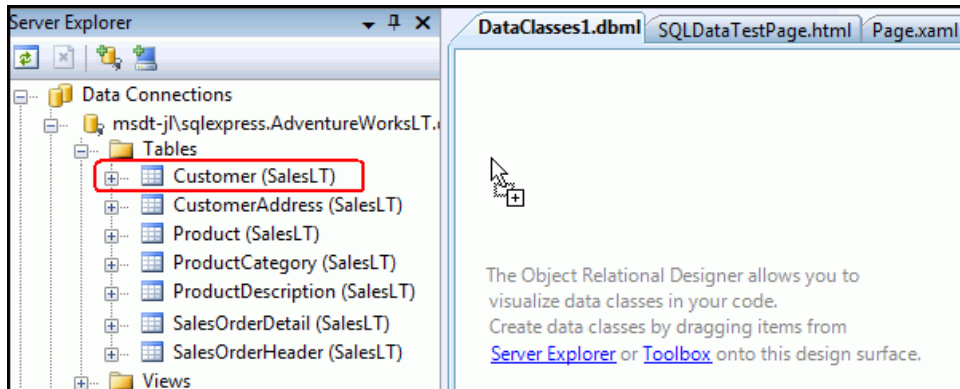


Figure 3-4. Dragging a table onto the Designer

Make the resulting LINQ class Serializable

While a LINQ class will be generated for you corresponding to the Customer table the default is for that class not to be serializable, but for it to usable in a web service we need to change that.. Click on the design surface to bring up the properties of the entire class, and set the Serializaition mode from None to Unidirectional

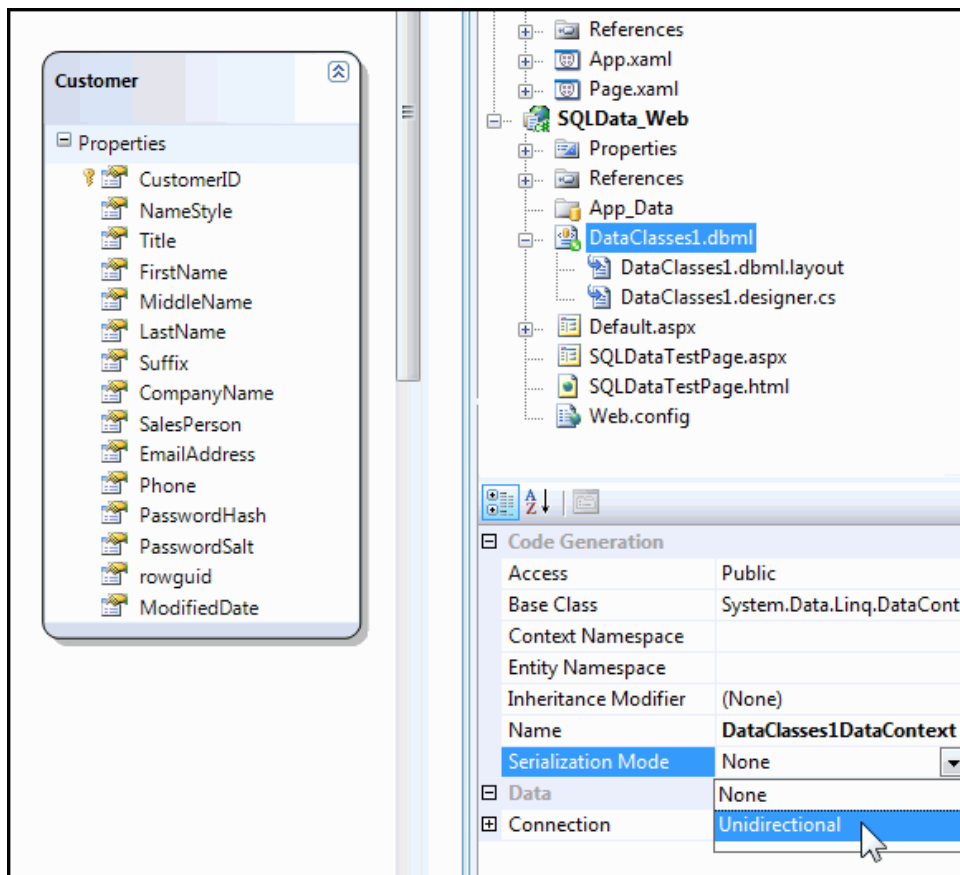


Figure 3-5. Making the Linq class serializable

Create the Web Service

You created the LINQ class (though not the query) first so that the web service (and Intellisense) will know about the Customer class and its members. With that in place we can ask Visual Studio 2008 to help create the Web Service.

Right click on the test project and choose Add New and from the templates choose WCF Service,

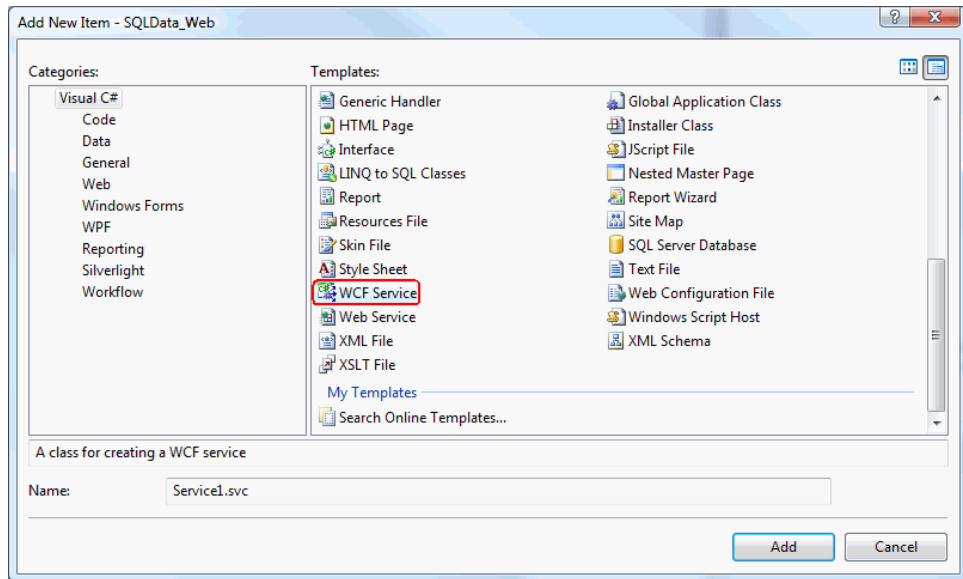


Figure 3-6. Creating a WCF Web Service

The result is the creation of three new files that hold the service contract for your WCF web service,

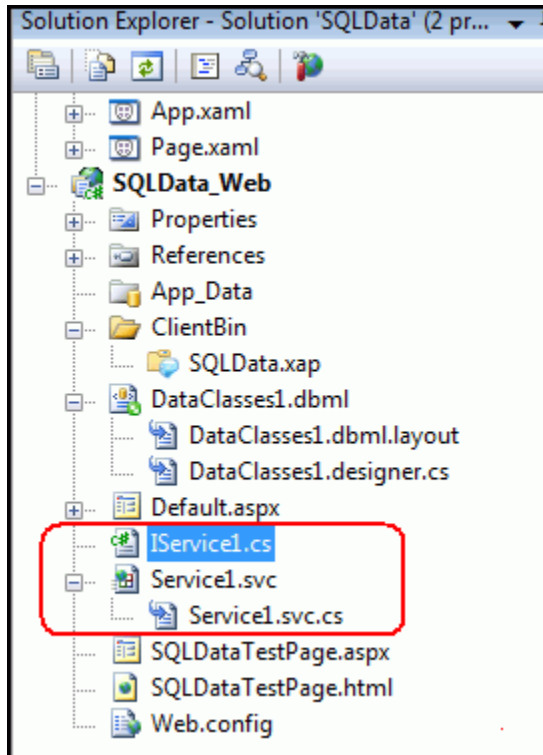


Figure 3-7. The WCF Web Service added to your project

Note that if you are more familiar with .NET 2 proxy based web services, you are in for a bit of an adjustment. WCF exposes more of the underlying XML and provides more of a contract-based programming model. You can find out more about WCF here: <http://tinyurl.com/ymxjkq>.

Open the first file, IService1.cs which contains the contract that was created by Visual Studio 2008.

```
public interface IService1
{
    [OperationContract]
    void DoWork();
}
```

We can replace this “dummy” contract with whatever contract we want our web service to provide. For the purposes of this tutorial we want to contract that our web service will return a list of Customer objects given a string that represents the beginning of a customer’s last name.

Thus, we’ll modify the method from returning void to returning List<Customer>.. However, as soon as you start to change the return type Intellisense is able to pop up to help you, specifically because we created this type in the LINQ class we defined earlier.

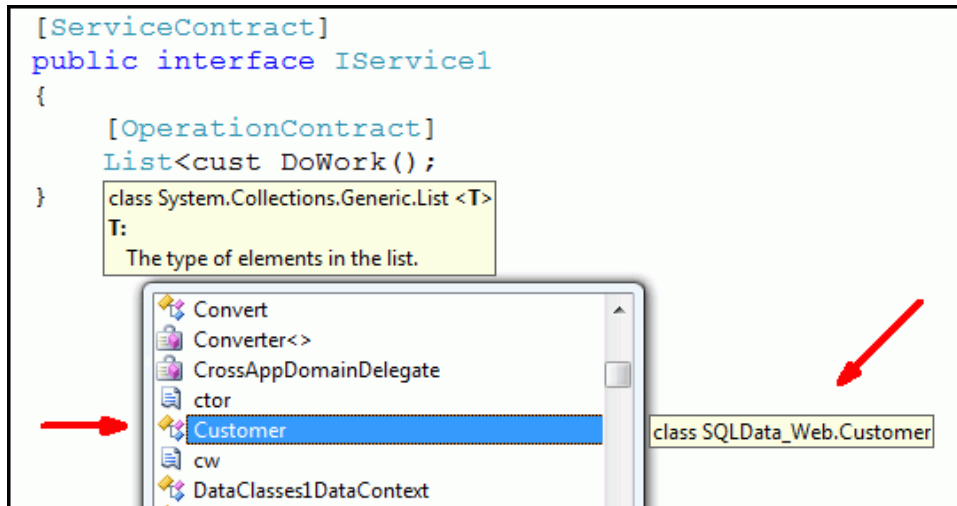


Figure 3-8. Intellisense is now aware of your Customer

The convention for a method that returns a set of foo given a bar is to name it GetFoosByBar. Thus, we'll name this GetCustomersByLastName.

```

public interface IService1
{
    [OperationContract]
    List<Customer> GetCustomersByLastName(string lastName);
}

```

Having changed the contract in the interface, you must be sure to change the implementation in the .cs file. But why work so hard? When you get to the cs file, just click on the interface and a smart tag will appear. Open the tag and it will offer to create the implementation skeleton for you!

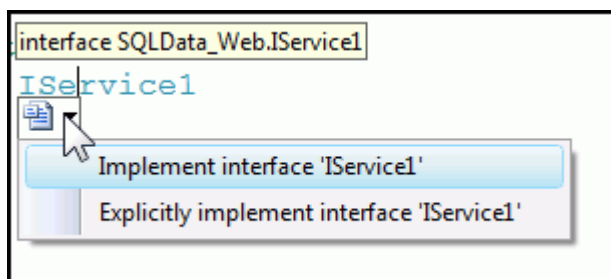


Figure 3-9. Intellisense can help implement the Interface

Throw away the DoWork method and fill in the GetCustomersByLastName with the LINQ query,

```

public class Service1 : IService1
{
    #region IService1 Members

```

```

public List<Customer> GetCustomersByLastName(string lastName)
{
    DataClasses1DataContext db = new DataClasses1DataContext();
    var matchingCustomers = from cust in db.Customers
                            where cust.LastName.StartsWith(lastName)
                            select cust;

    return matchingCustomers.ToList();
}

#endregion
}

```

(The region comments were put in when I asked the smart tag to create the implementation skeleton)

LINQ Syntax

Ah, finally, some LINQ. The trick to learning LINQ is to find [Anders](#) and have him explain it to you. No one does it better. Failing that, read [Scott's tutorial](#), or the chapters on LINQ in my [C# 3.0 book](#), or one of the many great books on LINQ.

Let's take this one LINQ statement apart. First, we use the new var inference variable which, surprisingly is type safe (it infers the type, but it is not without type!). We assign to it the result of the LINQ query, which will be an object of type IEnumerable.

The query syntax is much like SQL except that the Select statement comes at the end. So, in English, "Give me a connection to the database I told you about earlier and name that connection db. Go into that database and find the table named customers and look for each record where the LastName field begins with the letters held in the string lastName. Give me all the matching records. Assign all those records to the object matchingCustomers, which is smart enough to (a) know that it has to be of type IEnumerable and (b) know that when I call ToList() on it it should return a List<Customer>.

Watch Out for the Binding!

WCF uses wsHttpBinding as its default binding, in the Web.config file,

```

<services>
  <service behaviorConfiguration="SQLData_Web.Service1Behavior"
           name="SQLData_Web.Service1">
    <endpoint address="" binding="wsHttpBinding"
              contract="SQLData_Web.IService1">
      <identity>
        <dns value="localhost"/>
      </identity>
    </endpoint>
  </service>
</services>

```

Silverlight, however, supports only basic binding (SOAP 1.1, etc.), so you will need to change the binding accordingly,

```
<endpoint address="" binding="basicHttpBinding" contract="SQLData_Web.IService1">
```

That's it. Your service is ready to go.

Creating the Silverlight Application

The next step is to create the Silverlight Application that will interact with this web service. To do so, right-click on the references in the Silverlight project and choose Add Service Reference

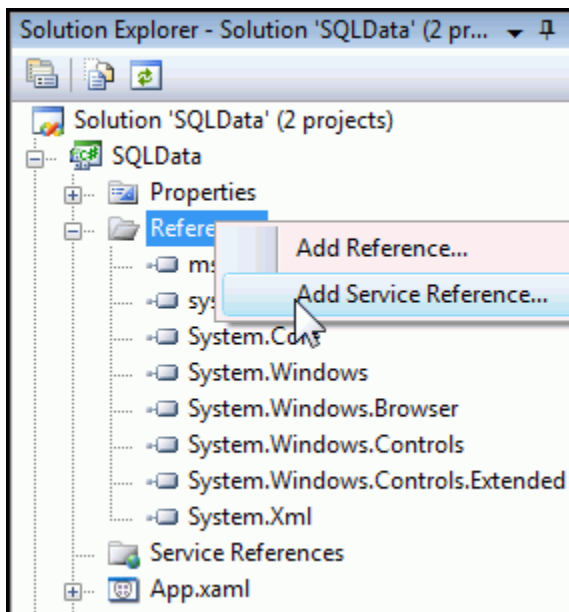


Figure 3-10. Adding a reference to the Web Service

When the Add Service Reference comes up click on Discover and choose **Services in Solution**. The service you created will be found. Before clicking OK notice that by clicking on the Service, the operation you created (GetCustoemrByLastName is discovered

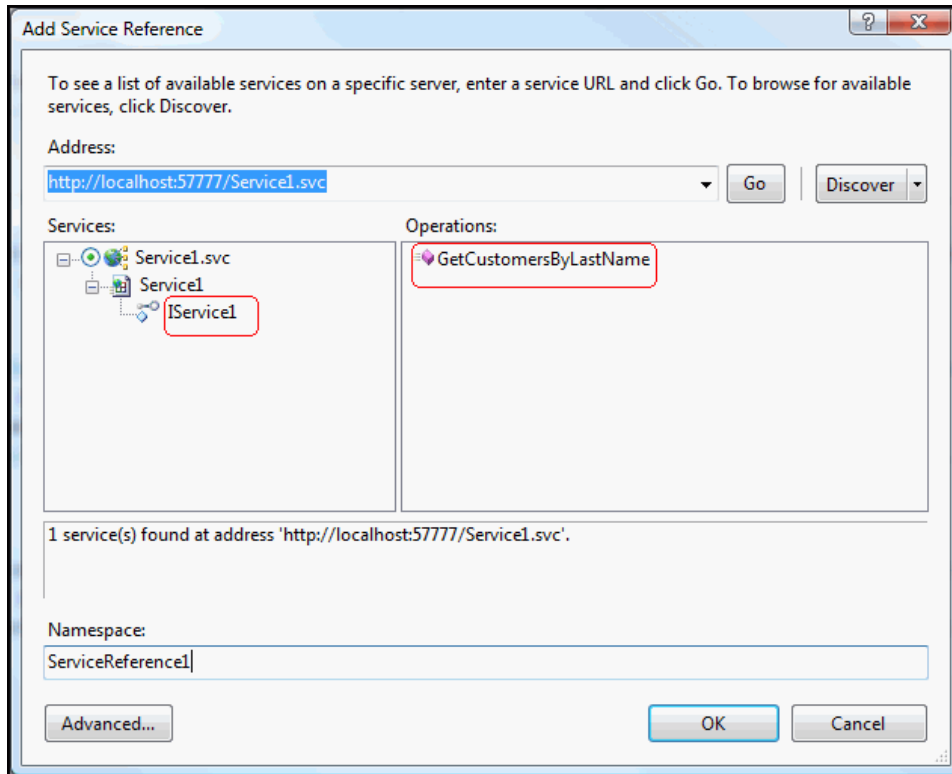


Figure 3-11. Choosing the operations you want to Add

Clicking OK adds the service to your project. You will access the Web Service (and its method) through this reference.

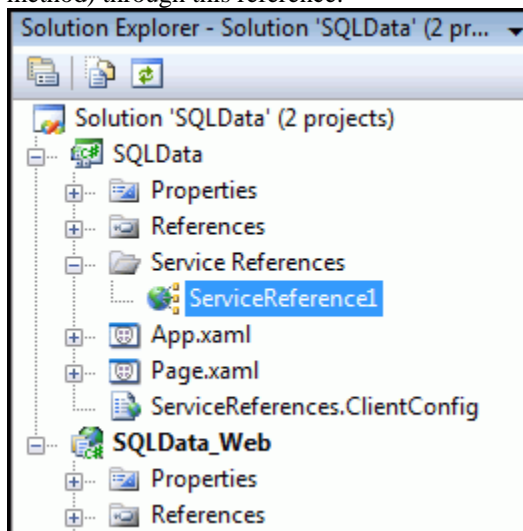


Figure 3-12. The reference added to your project

Creating the XAML

In the Page.xaml I'll create a very simple UI that will consist of a top row to enter the user's last name and a bottom row to display the results. To start, I'll layout the Grid's rows and columns,

```
<Grid x:Name="LayoutRoot" Background="White" ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition Height="10" /> <!--0 Margin-->
    <RowDefinition Height="50" /> <!--1 Prompts-->
    <RowDefinition Height="*" /> <!--2 DataGrid-->
    <RowDefinition Height="10" /> <!--3 Margin-->
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="10" /> <!--0 Margin-->
    <ColumnDefinition Width="*" /> <!--1 Controls-->
    <ColumnDefinition Width="10" /> <!--2 Margin-->
  </Grid.ColumnDefinitions>
</Grid>
```

Notice that I've set ShowGridLines to true while I'm working to ensure that I'm getting the results I hope for, and that the third row and second column use star sizing; indicating that they should take up all the remaining space.

The Grid has small margins on all sides and two rows, a top small row and a very large bottom row,

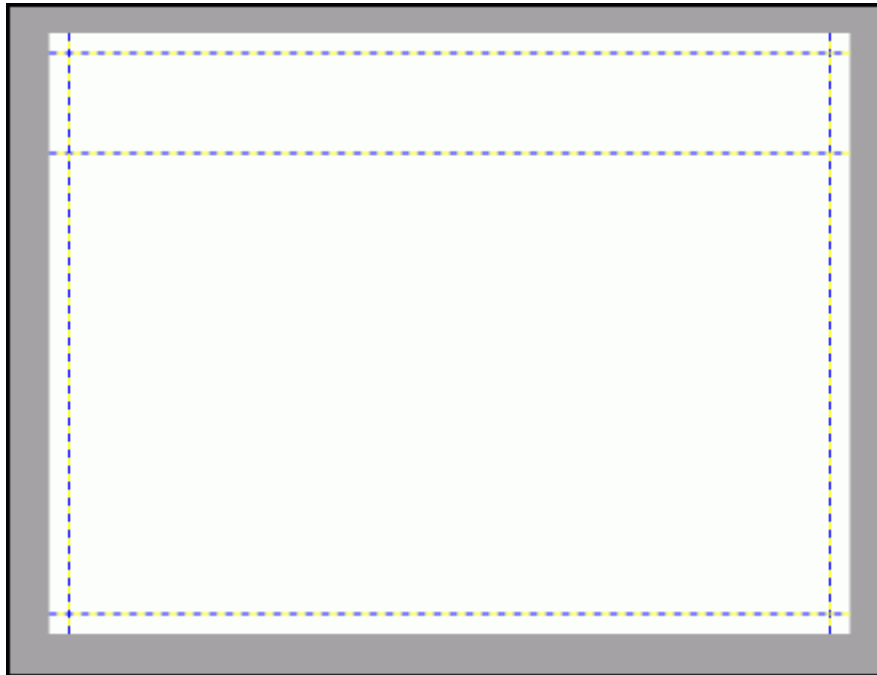


Figure 3-13. The grid in design mode

Placing controls in the top row

I want to place a Textblock (for the prompt) and TextBox (for input) and a button in the top row. The easiest way to do so is with a stack panel, and I'll surround it all with a border to set it off from the results.

```
<Border BorderBrush="Black" BorderThickness="2" Grid.Row="1" Grid.Column="1"/>
<StackPanel Grid.Row="1" Grid.Column="1" Orientation="Horizontal">
  <TextBlock Text="Last name to search for: " VerticalAlignment="Bottom"
    FontSize="18" Margin="15,0,0,0" />
  <TextBox x:Name="LastName" Width="250" Height="30" Margin="2,0,0,4"
    VerticalAlignment="Bottom"/>
  <Button x:Name="Search" Width="75" Height="30"
    Margin="20,0,0,4" Content="Search"
    VerticalAlignment="Bottom" Background="Blue" FontWeight="Bold"
    FontSize="14" />
</StackPanel>
```

Finally, drag a DataGrid from the Toolbox onto the XAML.

```
<my:DataGrid x:Name="theDataGrid" AlternatingRowBackground="Beige"
  AutoGenerateColumns="True" Width="700" Height="500" Grid.Row="2" Grid.Column="1"
  CanUserResizeColumns="True" />
```

You'll notice that it is given the prefix my and that a new namespace is declared to support it,

```
xmlns:my="clr-
  namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
```

Write the Event Handler for the Search Button

When the user clicks the search button we want to pick up the text in the Text box and give it to the web service, and get back a collection of customers. Let's set up the boilerplate event handling code in page.xaml.cs,

```
public Page()
{
  InitializeComponent();
  Loaded += new RoutedEventHandler(Page_Loaded);
}

void Page_Loaded(object sender, RoutedEventArgs e)
{
  Search.Click += new RoutedEventHandler(Search_Click);
}

void Search_Click(object sender, RoutedEventArgs e)
{
}
}
```

Call the Service Asynchronously

The only way to call a web service from Silverlight is asynchronously (which is fair as it is running in a browser and can't afford to block!)

The first task is to get a reference to the Web Service's Service1Client member. You can examine this in the object browser to see that it is this object that has the Asynchronous methods we'll need,

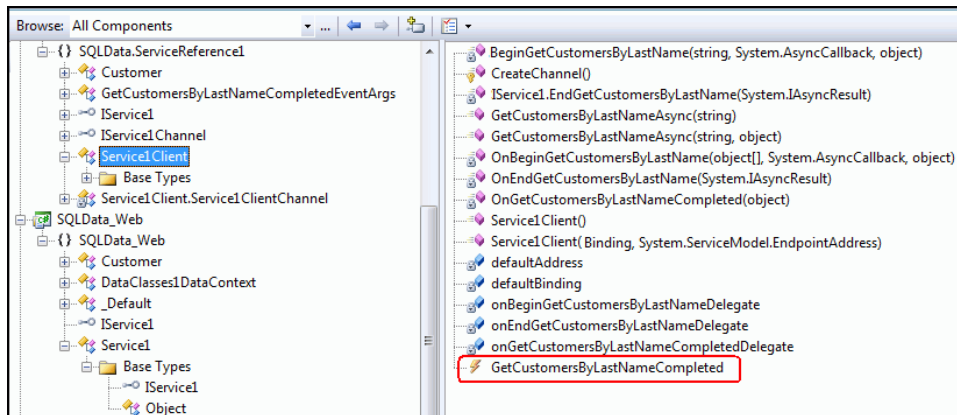


Figure 3-14. The Web Service client seen in the object browser

(image slightly abridged to save space)

We assign the Service1Client to the local object webService,

```
void Search_Click(object sender, RoutedEventArgs e)
{
    ServiceReference1.Service1Client webService =
        new SQLData.ServiceReference1.Service1Client();
}
```

We then use webService to set up an event handler for the method that will be called when the GetCustomersByLastNameCompleted event is called

```
webService.GetCustomersByLastNameCompleted +=
    new EventHandler<SQLData.ServiceReference1.
    GetCustomersByLastNameCompletedEventArgs>
    (webService_GetCustomersByLastNameCompleted);
```

Finally, we make the asynchronous call

```
webService.GetCustomersByLastNameAsync(LastName.Text);
}
```

When the service completes, the GetCustomersByLastNameCompleted event is raised, and our method is invoked. The carefully constructed list of Customers is stashed in e.Result which we assign to the DataGrid's ItemSource property and all the bindings now have a source to bind to.

```
void webService_GetCustomersByLastNameCompleted(
    object sender,
    SQLData.ServiceReference1.GetCustomersByLastNameCompletedEventArgs e)
{
    theDataGrid.ItemsSource = e.Result;
}
```

Test Page For SQLData

Last name to search for:

	LastName	Su	CompanyName	SalesPerson	EmailAddress	Phone	PasswordHash	Password
▶	Liu		Catalog Store	adventure-works\sn	david20@adventu	440-555-0132	61zeTkO+ej5g8GG0sw	c7Tlvv0=
	Liu		Chic Department S	adventure-works\j	jinghao1@adventu	928-555-0116	laD5AeqK9mRilrJi/eU	p6p0qKc
	Liu		Eastside Departme	adventure-works\li	kevin5@adventure	926-555-0164	yITpkiOHKLCjihNJ50j/	TgZnU0g
	Logan		Cycle Merchants	adventure-works\g	todd0@adventure-	783-555-0110	FV6z03ywwJOUmcU+	mFRhaEg
	Looney		Fitness Hotel	adventure-works\j	sharon2@adventu	377-555-0132	Uo3kAuNh936QPtIFP	uHgb0IU-
	Los		Healthy Activity St	adventure-works\li	jeremy0@adventur	911-555-0165	JLMkpmNutZfZw7s5	JK9/WX8
	Leavitt		Frugal Bike Shop	adventure-works\si	etsa0@adventure-v	482-555-0174	BmJaM+147GrhU00kN	YADhpPo
	Lawrence		Gear-Shift Bikes Li	adventure-works\j	david19@adventur	653-555-0159	HyZexVTTLkFfx/Sb+	/kc6RdY
	Lucerne		Grand Industries	adventure-works\g	anita0@adventure-	164-555-0118	YYTWMHbz9XZM2XoY.	Zb601a0
	Laszlo		Instruments and Pe	adventure-works\j	rebecca2@adventu	1 (11) 500 555-	ox4SPBzzVPKyCVolcZ	n7ydrcc=
	Lang		Kickstands and Acc	adventure-works\p	eric6@adventure-v	932-555-0163	katp5sn21zq5Z26YI	xv9RymE
	Lundahl		Leading Sales & Re	adventure-works\j	judy1@adventure-	260-555-0130	VzG/BDJkh2mMWAn2	N5Ajt+s=
	Lunt		Main Bicycle Servic	adventure-works\li	sean4@adventure-	183-555-0111	NCIEHFdWOrgFDd65	ek6WzwK
	Lepro		More Bikes!	adventure-works\li	bonnie2@adventur	354-555-0130	QFcESEnv3fd3s0Kxrfta	WVm7PPp
	Leste		National Manufact	adventure-works\g	linda7@adventure-	493-555-0134	heRwLFQMae6Y0X7+C	OHu+9rc
	Lewin		Town Industries	adventure-works\j	elsie0@adventure-	803-555-0116	sbrtAXJY79C5nTFNakt	UUwXzgY
	Li		Security Racks and	adventure-works\si	george3@adventur	699-555-0183	y3Yyy5PHg7L/+LZkIF	4m5UYgA
	Li		Rapid Bikes	adventure-works\j	yale0@adventure-	316-555-0138	B2RR480ridZsqURXY3	5MEUKYL
	Li		Nearby Sporting Gc	adventure-works\j	yuhong1@adventur	1 (11) 500 555-	XrcuygOte7eGdTleJO	R273GBA
	Lique		Front Sporting Goo	adventure-works\p	joseph2@adventun	119-555-0195	7dnLSdRuWFKBMjmOI	JH5dJh4+

Figure 3-15. The Running Program

Hey presto!

Once you know how, the effort to make this kind of application, even using Styles and more to make it look just the way you want, is measured in hours, at most.