DevForce and Report Sharp-Shooter Silverlight Bundle

Getting Started

This document walks you through the process of adding reporting capabilities to a DevForce Silverlight application, using PerpetuumSoft's *Report Sharp-Shooter for Silverlight* product.

The starting point for this walk-through is the completed application from the *Four Simple Steps to Your DevForce Silverlight app* tutorial, available in the Learning Resources for DevForce.

In this walk-through, you'll see how to create and configure a report service, design a report, and integrate a client-side report viewer into a Silverlight application page.

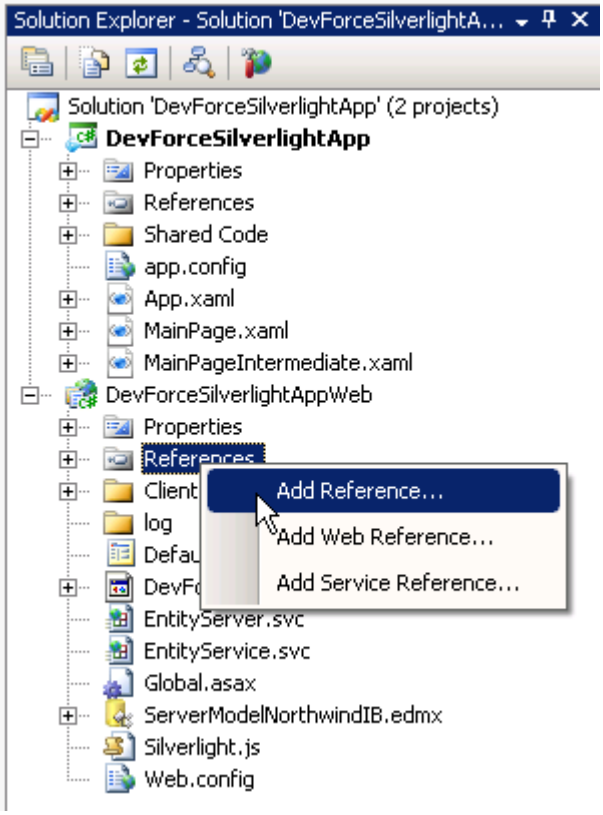
Software Prerequisites

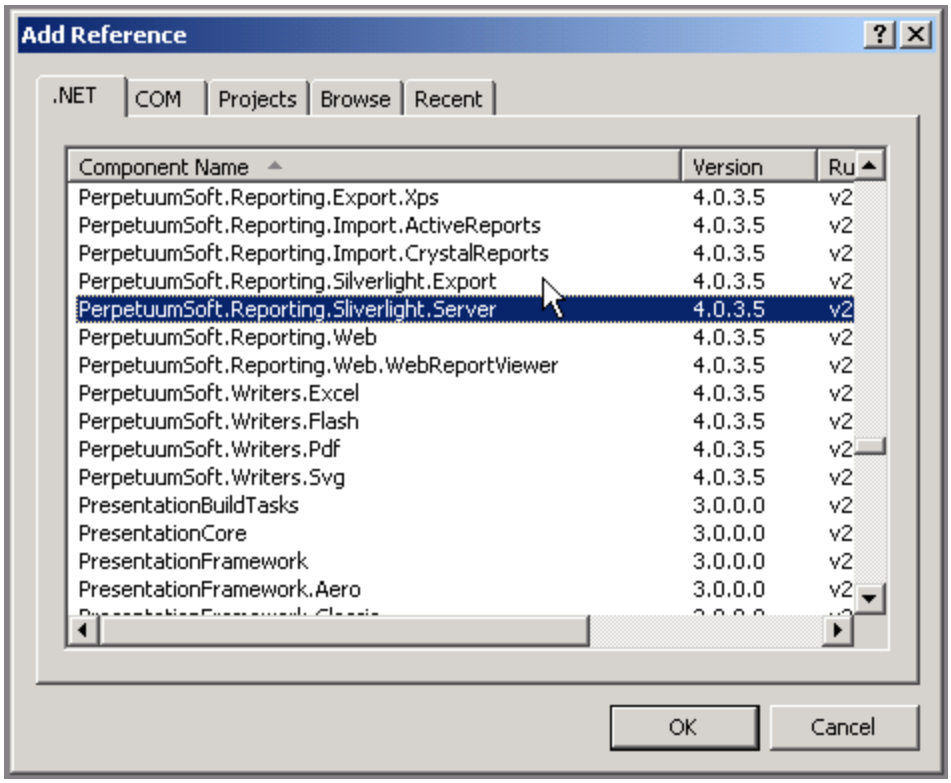
In order to get the sample working the following software should be installed on your computer:

- MS Visual Studio 2008
- .Net 3.5
- ASP.Net 2.0
- Silverlight 3.0
- Report Sharp Shooter for Silverlight 4.0.3.0 or higher.
- DevForce 2009 Silverlight, version 5.2.4 or later

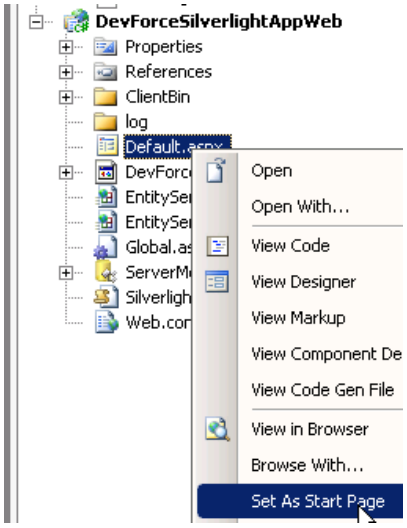
Step 1. Add and Set Up a WCF Service for Report Sharp-Shooter

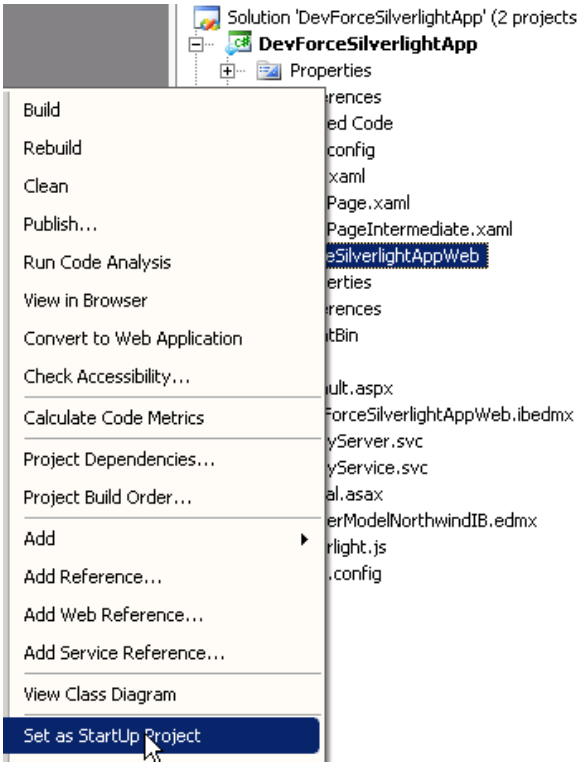
In order to permit your client application to interact with the server to produce a report, we need a special web service. This service will need a reference to the assembly *PerpetuumSoft.Reporting.Silverlight.Server.dll*. Add this by right-clicking the References node of the *DevForceSilverlightAppWeb* project and choosing *Add Reference* from the popup menu.



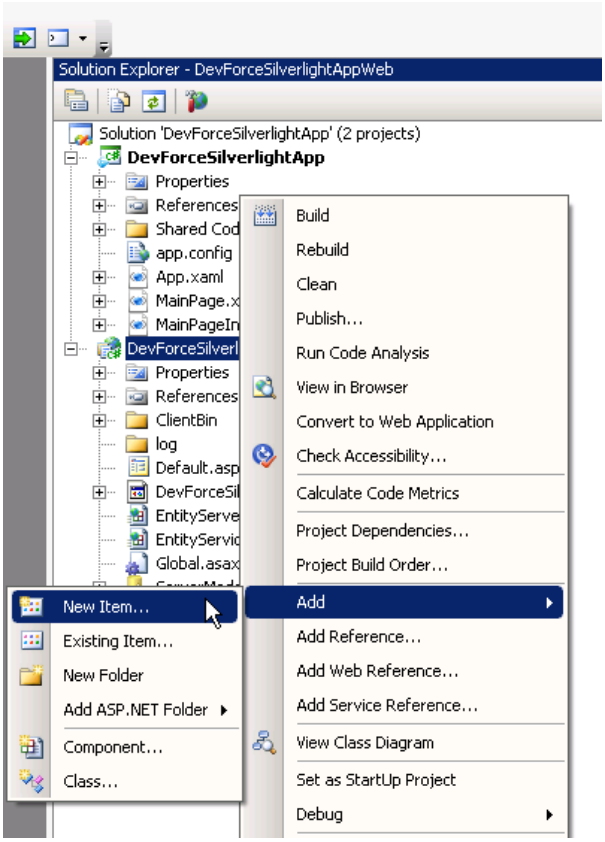


While we're here, let's make *DevForceSilverlightAppWeb* the solution's Start Project (shown at right), and *Default.aspx* that project's Start Page (below).

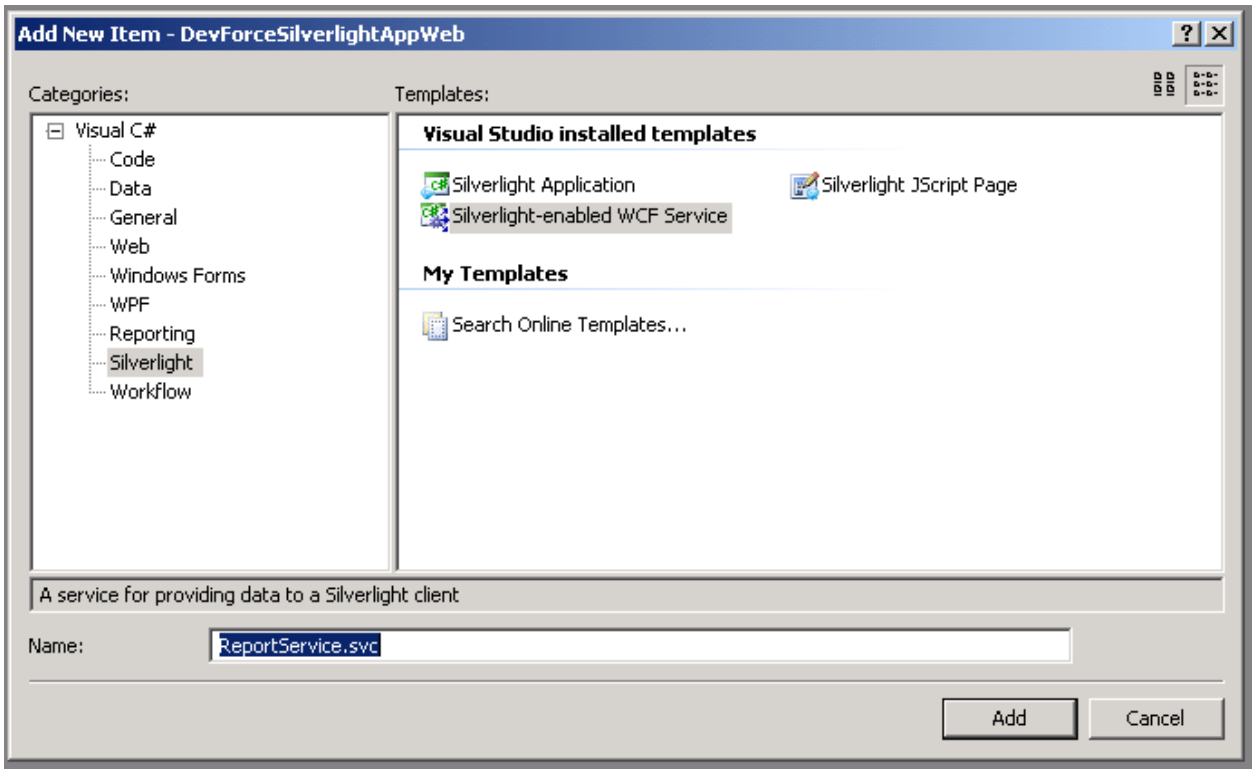




Add a Silverlight-enabled WCF Service named ReportService.svc to the *DevForceSilverlightAppWeb* project. To do that, right-click the *DevForceSilverlightAppWeb* node and choose Add->New Item from the popup menu.



Add the Silverlight-enabled WCF Service template to your project.



When the source code opens for the ReportService class (ReportService.svc.cs), add the following *using* statement at the top:

C#

```
using PerpetuumSoft.Reporting.Silverlight.Server;
```

Replace the generated code inside (and inclusive of) the namespace statement block with the following code:

C#

```
namespace DevForceSilverlightApp {
  [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Required)]
  public class ReportService : ReportServiceBase {
    ReportService() {
      InitializeComponent();
    }

    private void InitializeComponent() {
    }
  }
}
```

VB

The *ReportService class* will extend the *PerpetuumSoft.Reporting.Silverlight.Server.ReportServiceBase* class and will contain an implementation of a WCF-service for the Silverlight ReportViewer.

> Note that our replacement code omits the *[ServiceContract(Namespace = “”)]* attribute from the *ReportService* class. That’s because the *ReportServiceBase* class implements this attribute.

Save and close the *ReportService.svc.cs* file.

Now you need to configure the created WCF service by making modifications to the system.serviceModel section in the *web.config* file.

> We’ll walk you through these changes individually, but for your reference we have included in *Appendix A* of this document a completed version of the *web.config* file with all prescribed changes.

If one does not already exist, add an endpointBehaviors section to the existing <behaviors> section with the content shown below:

XML

```
<system.serviceModel>
  ...
  <behaviors>
    ...
    <endpointBehaviors>
      <behavior name="webBehavior">
        <webHttp/>
      </behavior>
    </endpointBehaviors>
    ...
  </behaviors>
  ...
</system.serviceModel>
```

In the <services> section of the web.config, find the definition of the created service:

XML

```
<service ... name="DevForceSilverlightApp.ReportService">
```

Find the endpoint with an empty address attribute (address=””):

XML

```
<endpoint address="" binding="customBinding"
   bindingConfiguration="customBinding0"
   contract="DevForceSilverlightApp.ReportService" />
```

Change the *binding* attribute value for that endpoint to “basicHttpBinding”, the *bindingConfiguration* attribute value to “”, and the *contract* attribute value to "PerpetuumSoft.Reporting.Silverlight.Server.IReportService" (or just replace the XML for this endpoint with the XML shown below):

XML

```
<endpoint address="" binding="basicHttpBinding"
   bindingConfiguration=""
   contract="PerpetuumSoft.Reporting.Silverlight.Server.IReportService"
/>
```

Add the following additional endpoint definition code to the definition of the service:

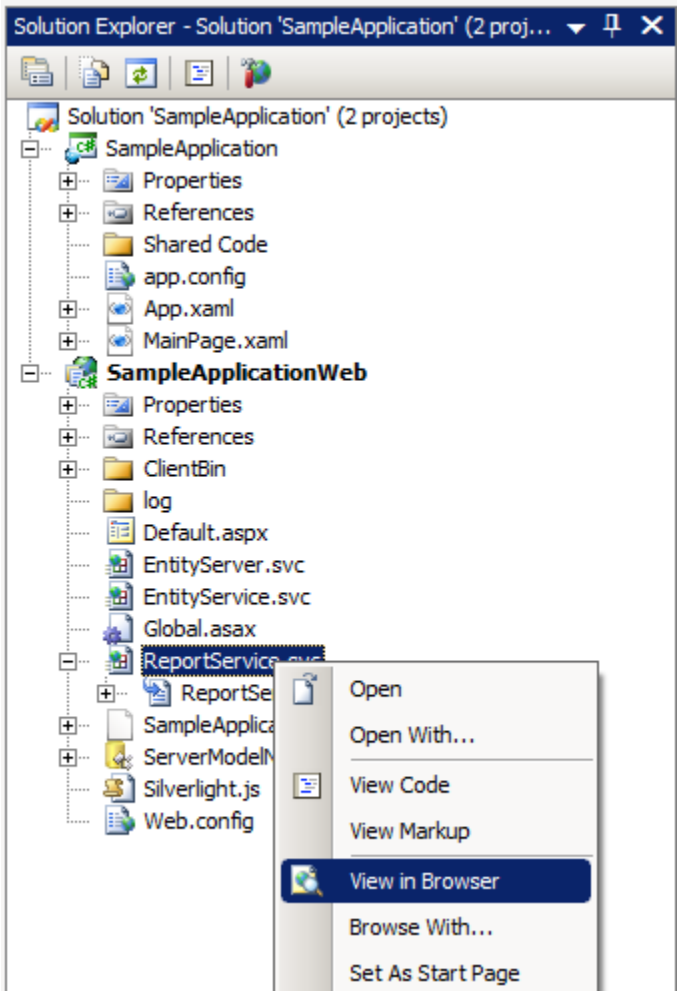
XML

```
<endpoint address="rest" behaviorConfiguration="webBehavior"
  binding="webHttpBinding"
  contract="PerpetuumSoft.Reporting.Silverlight.Server.IReportServiceResources"
/>
```

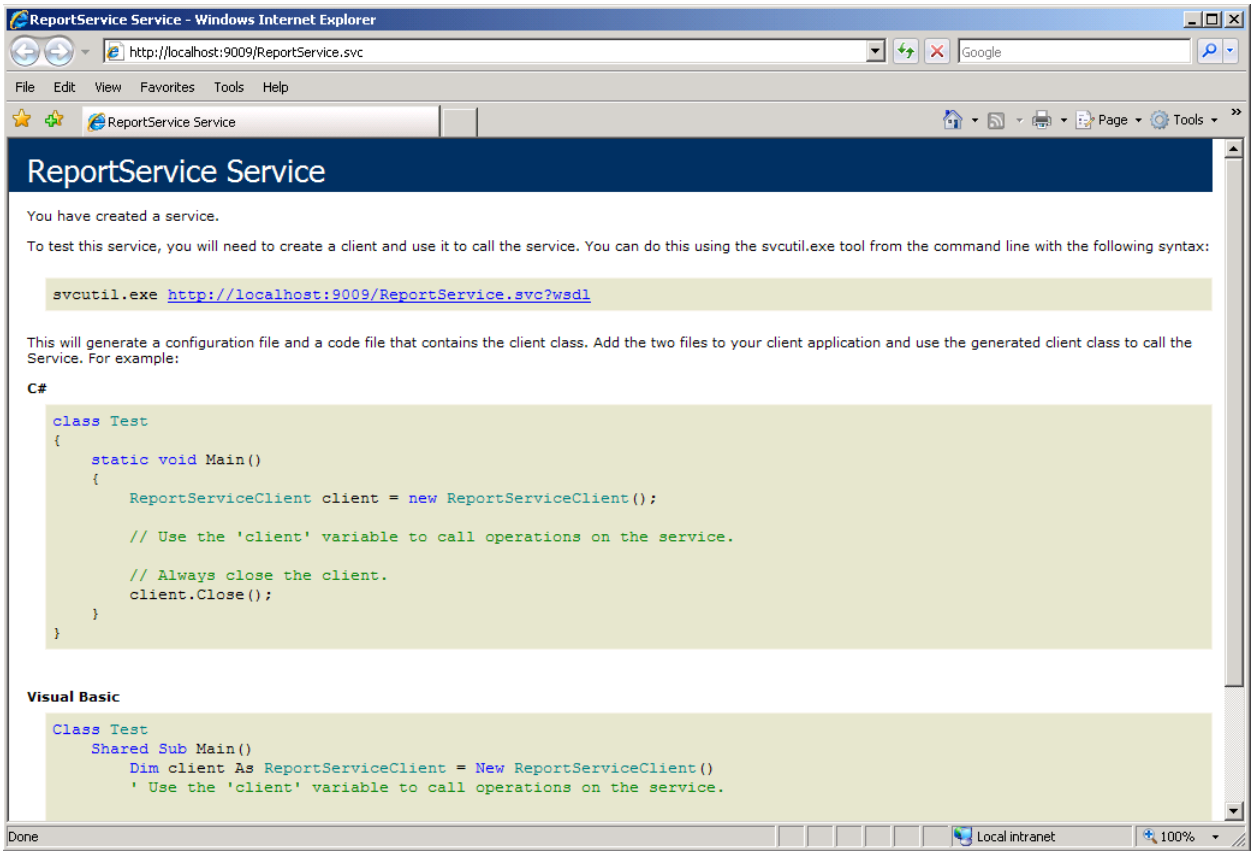
The uses of the various endpoint addresses is as follows:

- address="" is used with the report; for example, to get pages;
- address="rest" is used with the resources; for example, to get localization;
- address="mex" is used to get meta information about the service.

Now, let's check if the service works. To do this, right-click the *ReportingService.svc* node in the Solution Explorer and choose the "View in Browser" option:



If you have done everything correctly, you see the following in the browser window:

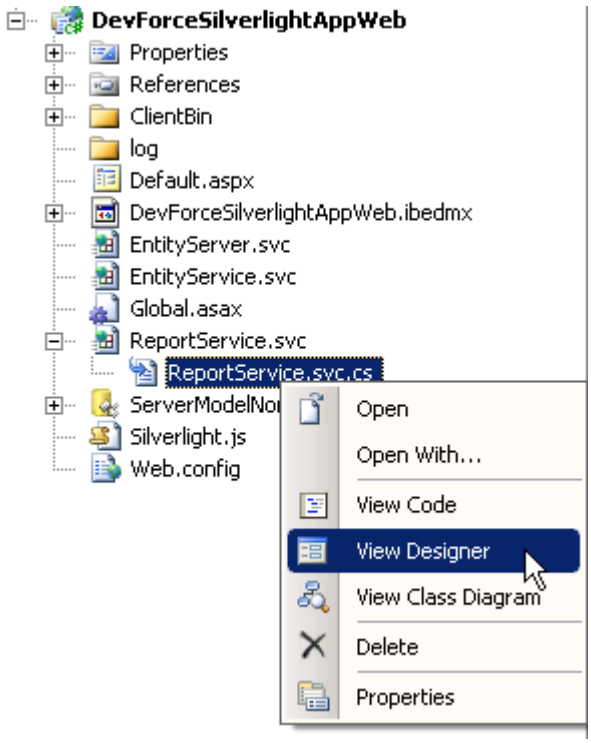


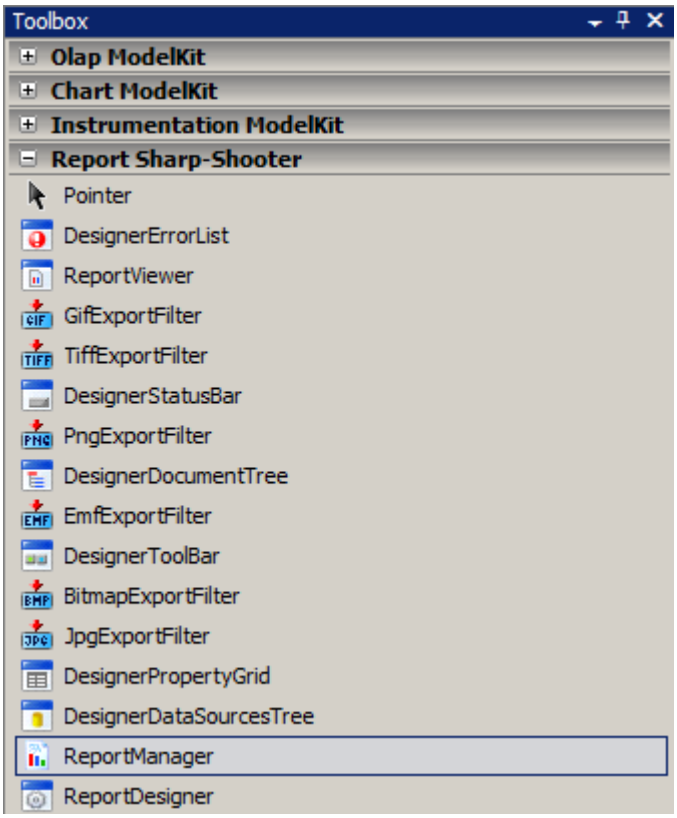
Close the updated *web.config* file.

Step 2. Add a ReportManager to the ReportService, Configure It, and Add Some Setup Logic for the Reports

To begin, you must designate a ReportManager component for our ReportingService. The ReportManager is responsible for report generation.

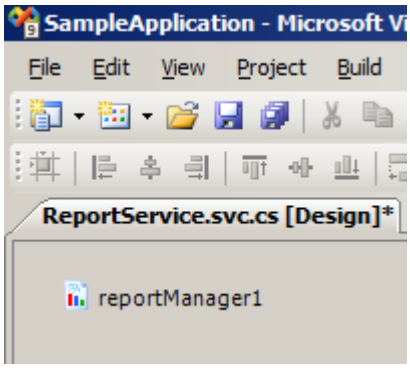
Open the service in the designer. To do that, right-click the *ReportService.svc.cs* file in the Solution Explorer and choose the *View Designer* item from the popup menu, as shown at right.

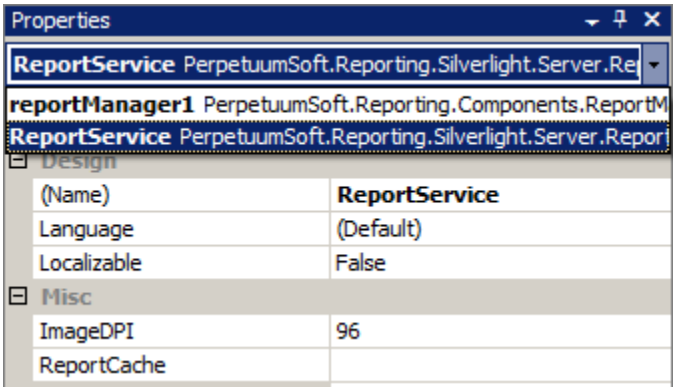




Now, add the ReportManager component (double click on ReportManager in ToolBox, as shown at left); this component is responsible for report generation.

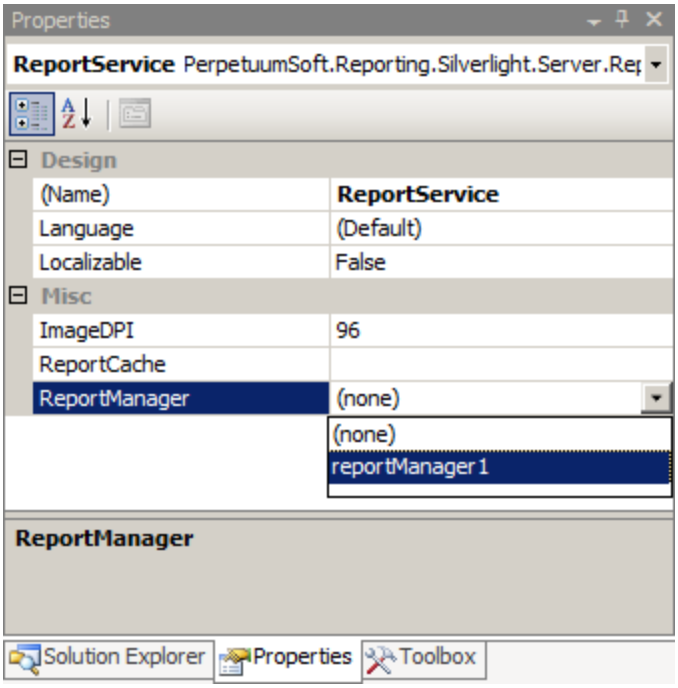
The *ReportManager* node (*reportManager1*) will appear in the designer:





In the drop-down list for the Properties window, find and select the *ReportService* component.

Find the *ReportManager* property for the *ReportingService*, and select *reportManager1* (the component you just dragged on to the designer surface for the *ReportService*) from the list:



The Data structures to be used by your reporting application will be obtained from the domain model you have already created; but you need to add them to the ReportManager's DataSources collection. Do this in code, in an override of the *ReportingService* class's *OnLoadData* method.

To view the latter, right-click the designer area of the ReportingService and click the "View Code" item from the contextual menu. Then add the following method to the ReportService class:

C#

```
protected override void OnLoadData(
  IDictionary<string, object> parameters,
  string reportName,
  PerpetuumSoft.Reporting.Components.ReportSlot reportSlot) {

  base.OnLoadData(parameters, reportName, reportSlot);

  switch (reportName) {
    case "EmployeesReport":
      IEntityQuery employeeQuery = _mgr.Employees
        .OrderBy(e => e.LastName)
        .ThenBy(e => e.FirstName);
      reportManager1.DataSources.Add("Employees", employeeQuery);
      break;

    case "CustomersReport":
      string headquartersCountry = (string)parameters["Country"];
      IEntityQuery customerQuery;
      if (headquartersCountry != null) {
        customerQuery = _mgr.Customers
          .Where(c => c.Country == headquartersCountry)
          .OrderBy(c => c.CompanyName);
      }
      else {
        customerQuery = _mgr.Customers.OrderBy(c => c.CompanyName);
      }
      reportManager1.DataSources.Add("Customers", customerQuery);
      break;

    default:
      throw new ArgumentException("Unknown report name");
  }
}
```

The code also uses a *DomainModelEntityManager* whose scope is the entire *ReportService* class, so add the following code to the bottom of the class:

C#

```
#region Private Fields
DomainModelEntityManager _mgr = new DomainModelEntityManager();
#endregion Private Fields
```

Finally, since the code references the *IEntityQuery* interface from the DevForce namespace *IdeaBlade.EntityModel*, you'll need a using statement for that namespace at the top of the file. You'll also need one for the *DomainModel* namespace that contains the *DomainModelEntityManager*:

C#

```
using IdeaBlade.EntityModel;
using DomainModelEntityManager;
```

Returning to the OnLoadData override, note that we are providing therein for two different reports: an "EmployeesReport" and a "CustomersReport". The "ExployeesReport", which we'll build as part of this tutorial, takes no parameters. The "CustomersReport" -- which we won't build but which is included in the code solution that accompanies this tutorial -- expects a "Country" parameter to be passed from the client. That will be used to filter the report output based on client-side input from the application's end user.

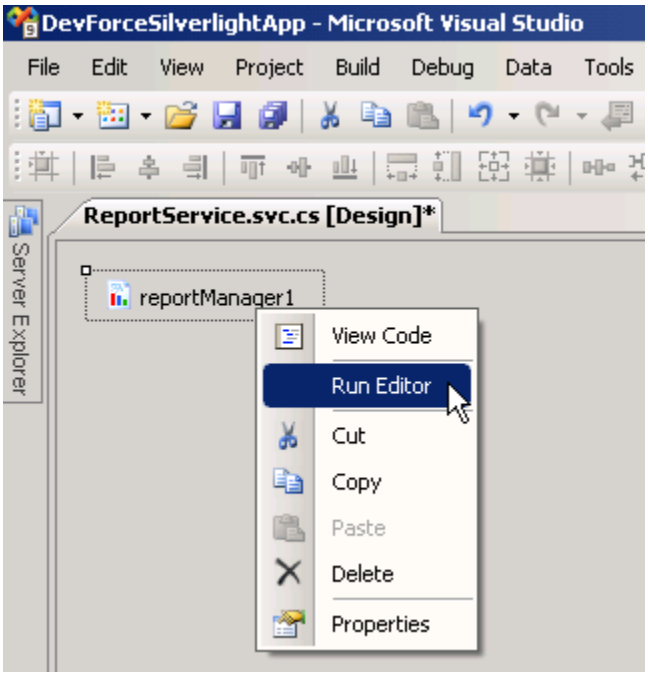
If you've made all the changes correctly, your Visual Studio solution will build at this point without errors. Try it!

Close all open code files, but leave the *ReportService* designer open.

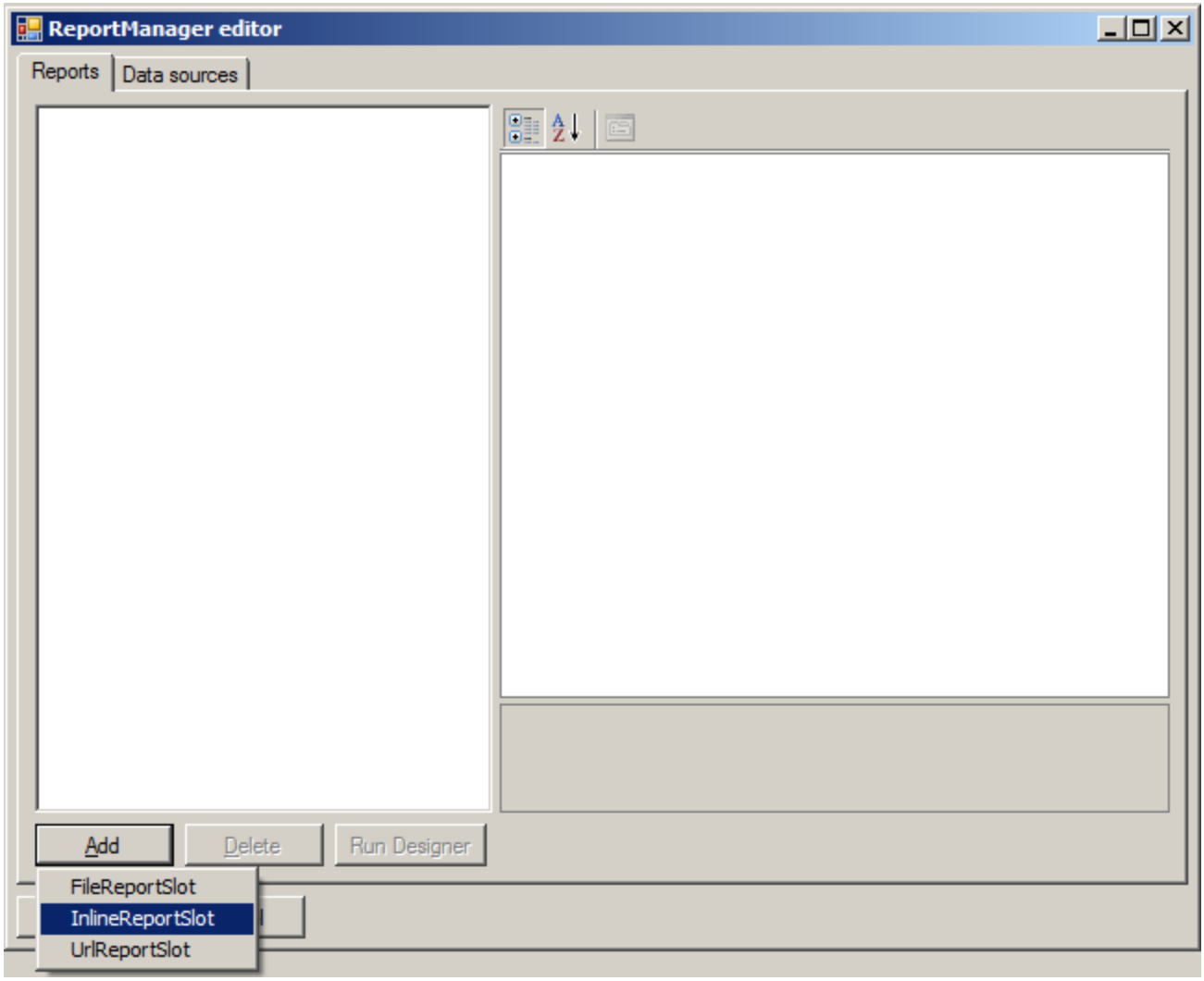
Step 3. Create the *Employee* Report Template

Now let's create a report.

Run the Report Manager Editor by right-clicking the reportManager1 component in the ReportingService designer and choosing the Run Editor option.



In the “Reports” tab, add a new object by clicking the Add button and selecting InlineReportSlot.[1]

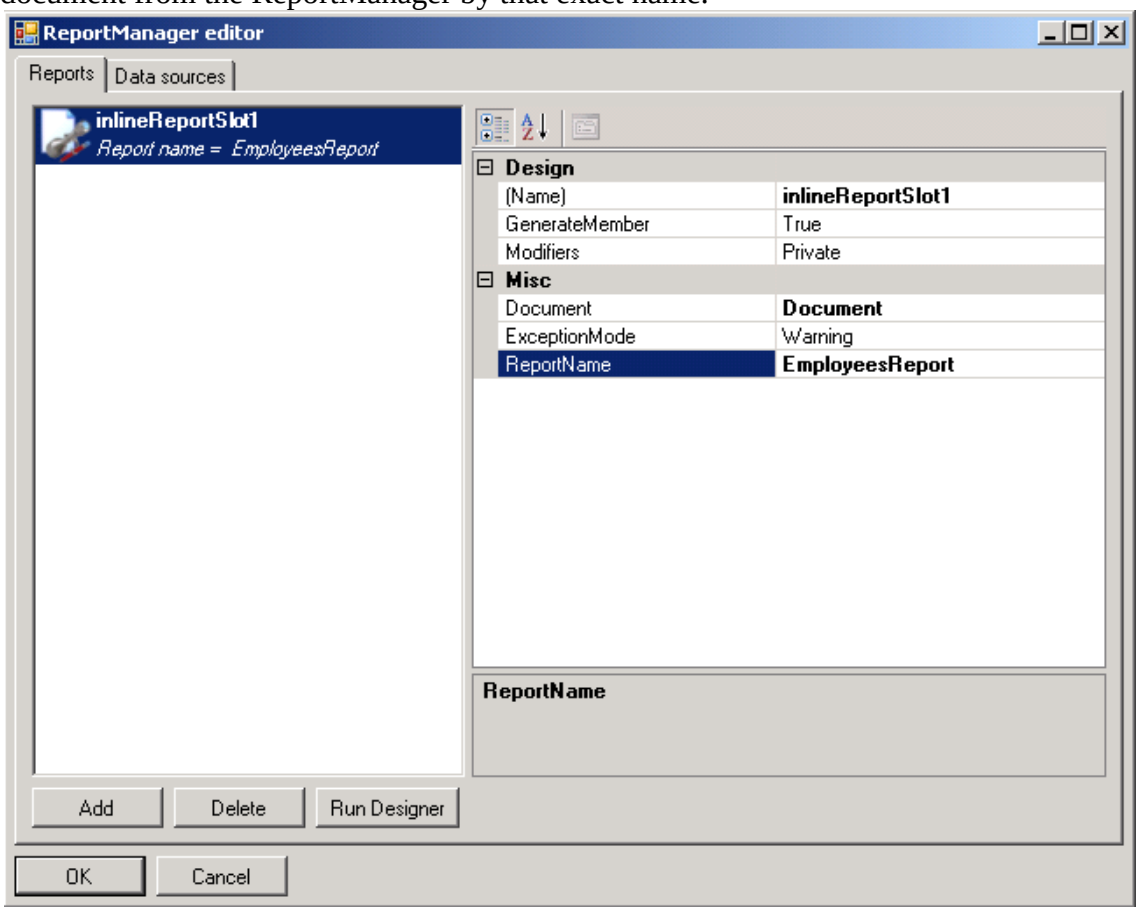


[1] A *FileReportSlot* saves a report template to a file. The path to the file is specified in the FilePath property.

An *InlineReportSlot* saves a template to the code of an application. So, the template cannot be changed without recompiling the whole application.

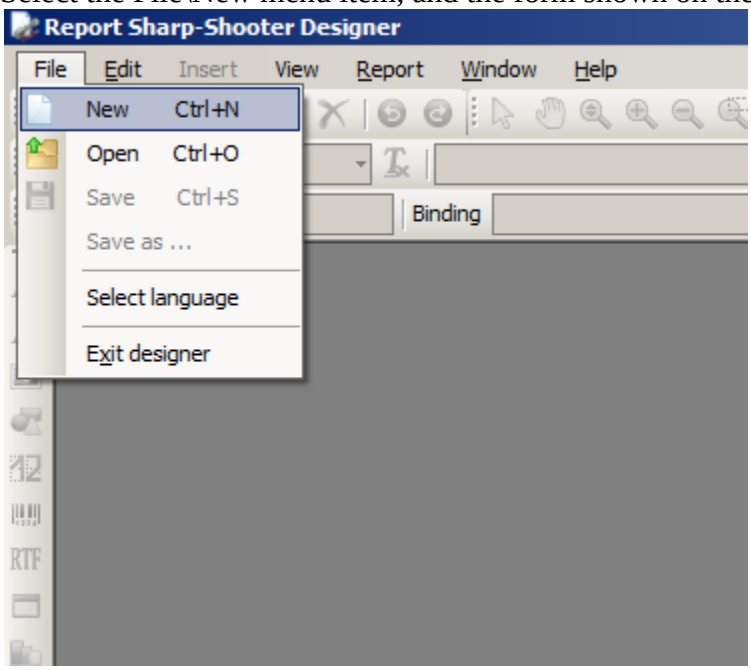
An *UrlReportSlot* saves a template to a resource by the indicated URL. And the URL is defined in the URL property.

Set the *ReportName* property value to “EmployeesReport”. Afterwards you will get the required document from the ReportManager by that exact name.

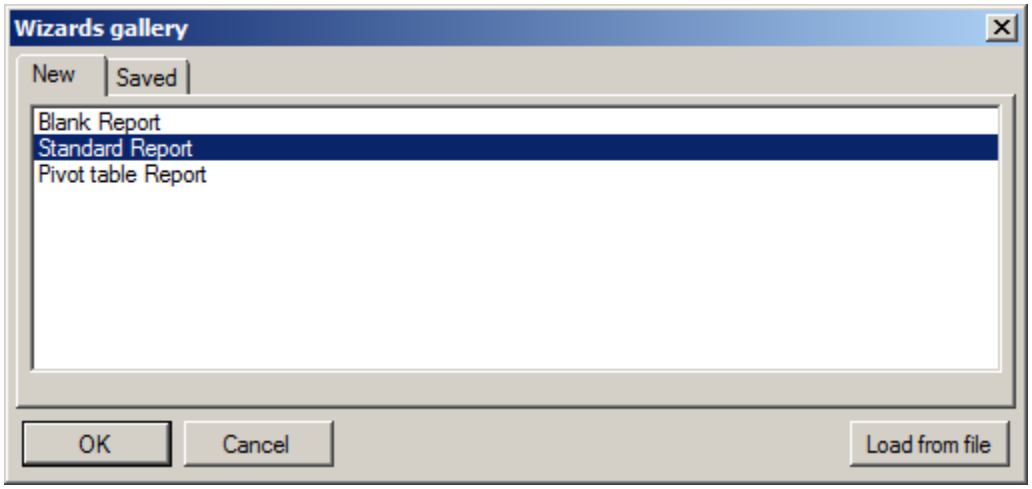


Now click the “Run Designer” button to launch the report designer.

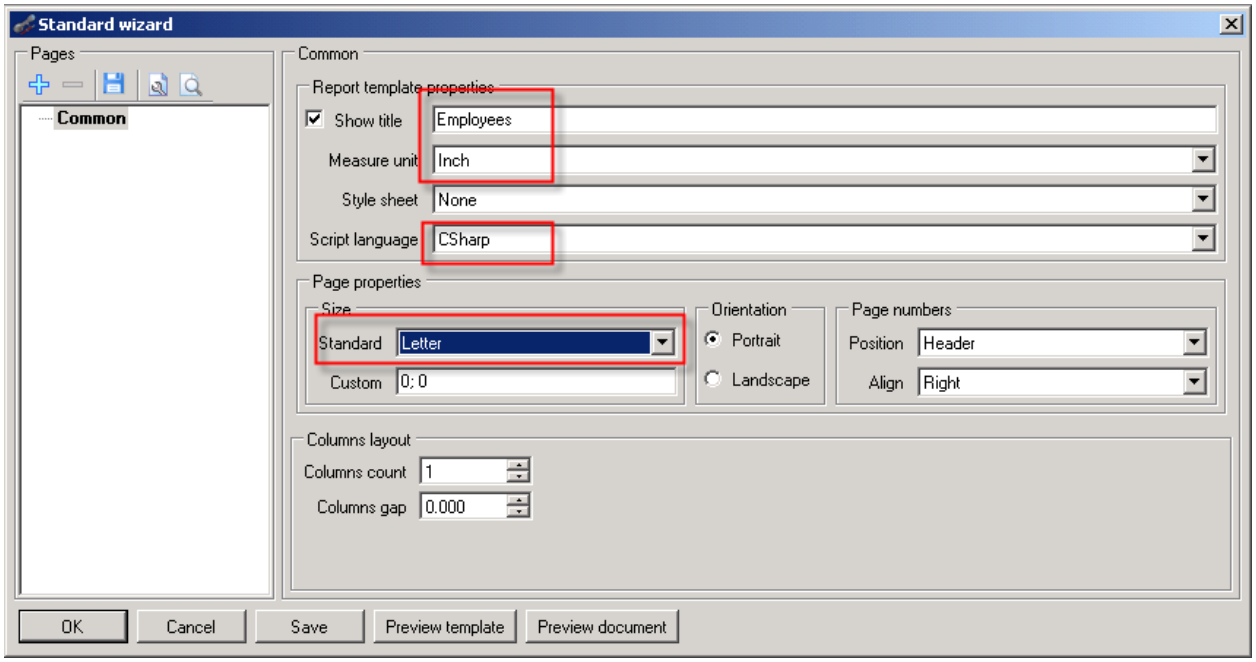
Select the File\New menu item, and the form shown on the screen below will appear.



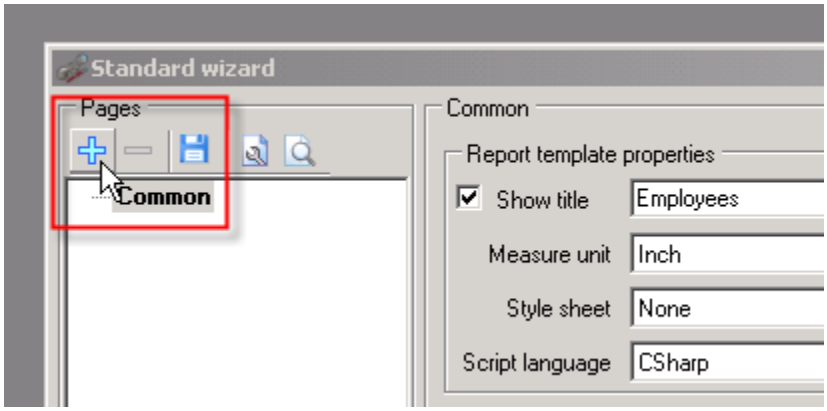
Select the Standard Report option from the list on the “New” tab and press OK.



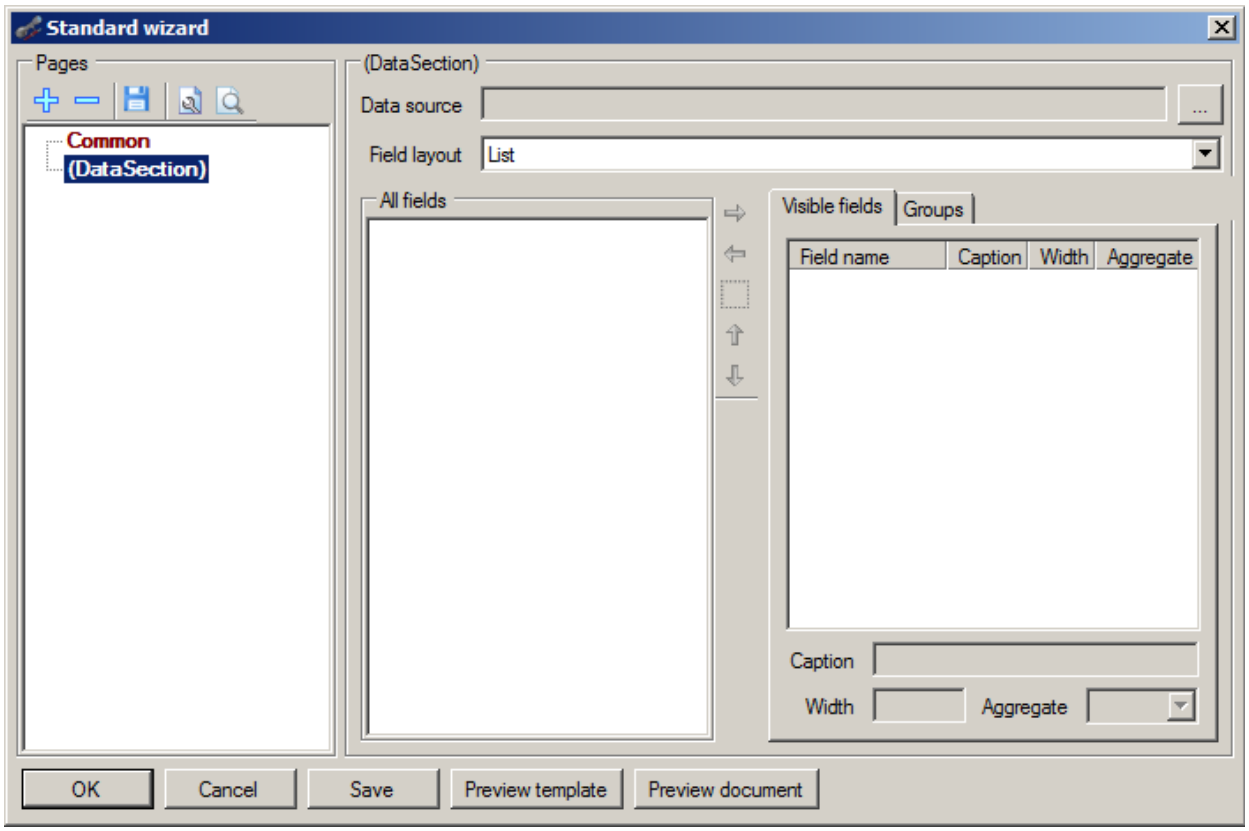
The Standard Wizard window will appear on the screen. Set document parameters as shown in the figure below:



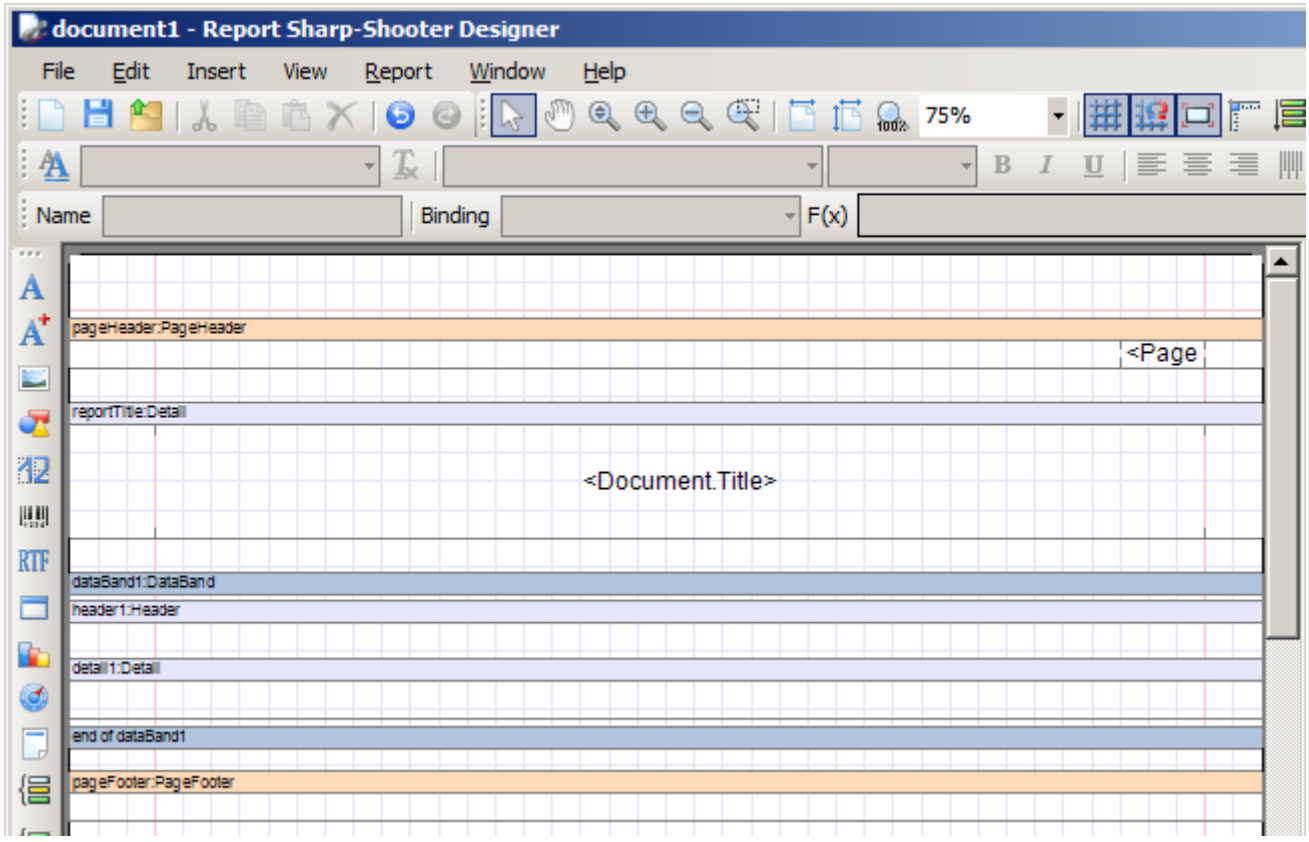
Add a data source using the Add button ( ).



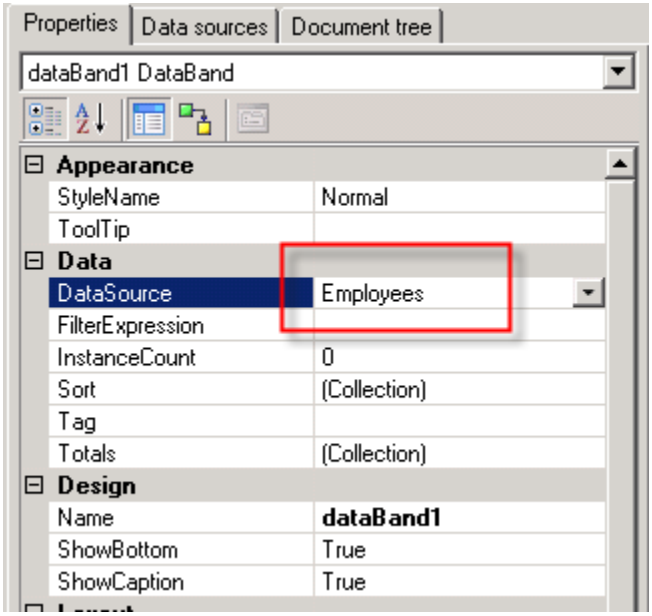
You should see the following dialog:



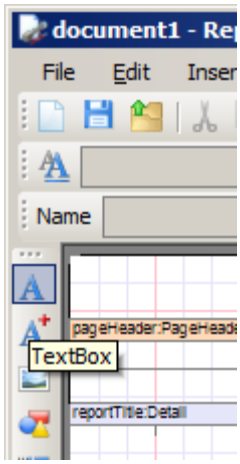
For this sample, since our data source is going to be created programmatically, there is no need to add anything further to this form. Simply click <OK>, and the report template is created:



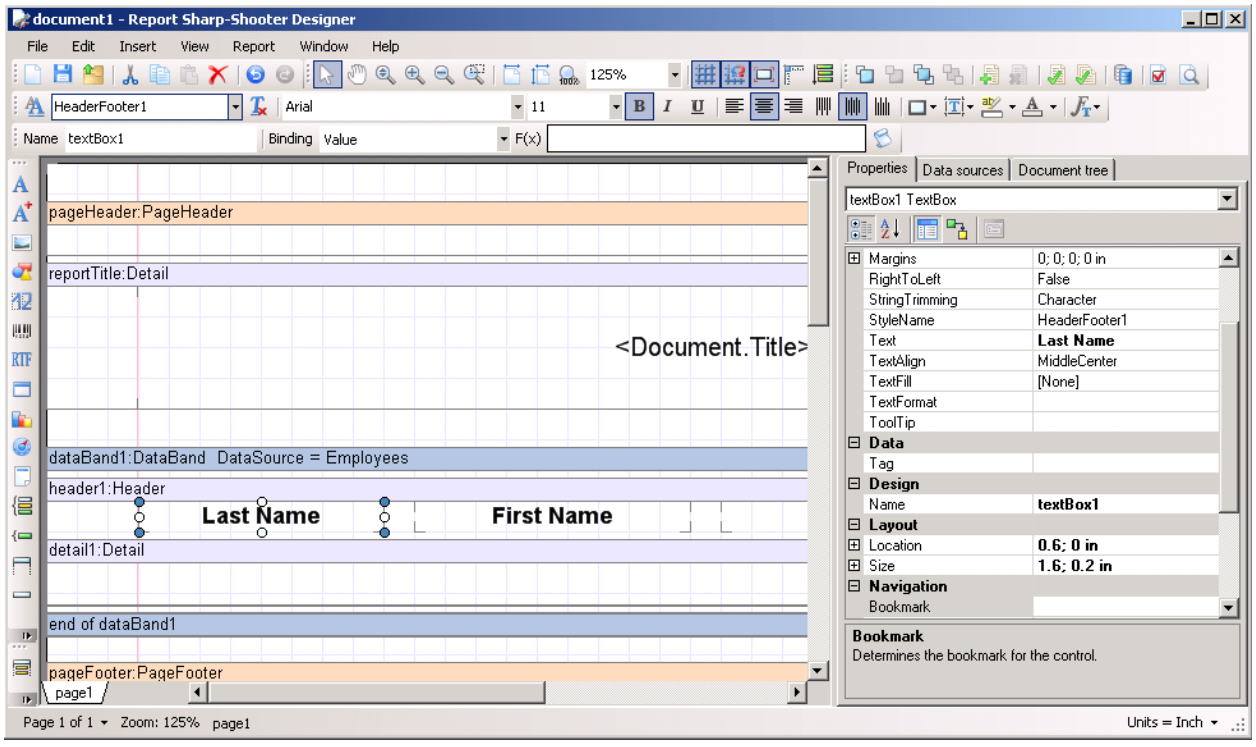
Select the dataBand1 component in the report designer. Set its DataSource property value to “Employees” by typing in that value. Again, because in this example the data source is created programmatically, we cannot pick its name from a list. Simply type in the name of the data source which you added to the ReportManager component in the beginning of the Step 2.



Add four TextBox controls to the header1 band in order to display column captions. To do this, select the TextBox button in the toolbar at the left of the report, then use your mouse to “draw” a textbox inside the header1 Header band. You will need to re-select the TextBox button in the toolbar before drawing each TextBox.

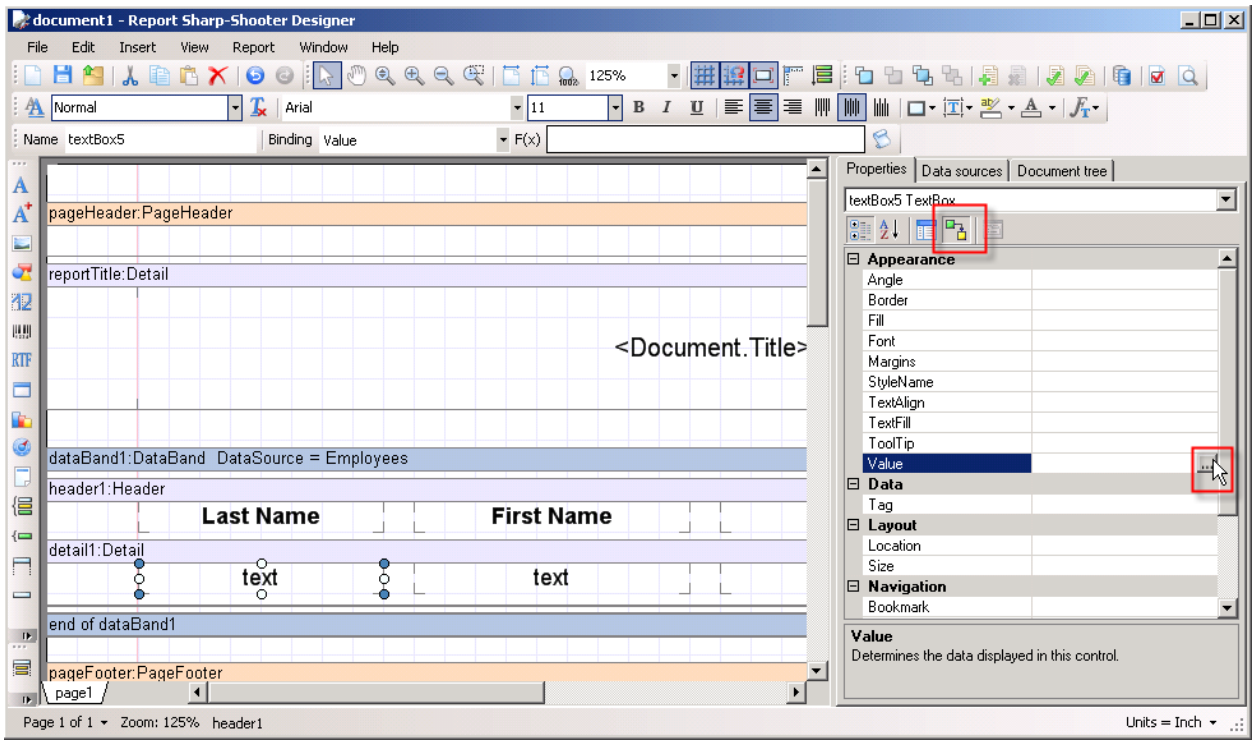


Set the *Text* property value of these Textboxes to “Last Name”, “First Name”, “Title”, and “Hire Date”, respectively. Select all four Textboxes by holding down the SHIFT key as you click on each. Set their font size to *11* and their style to *Bold*.

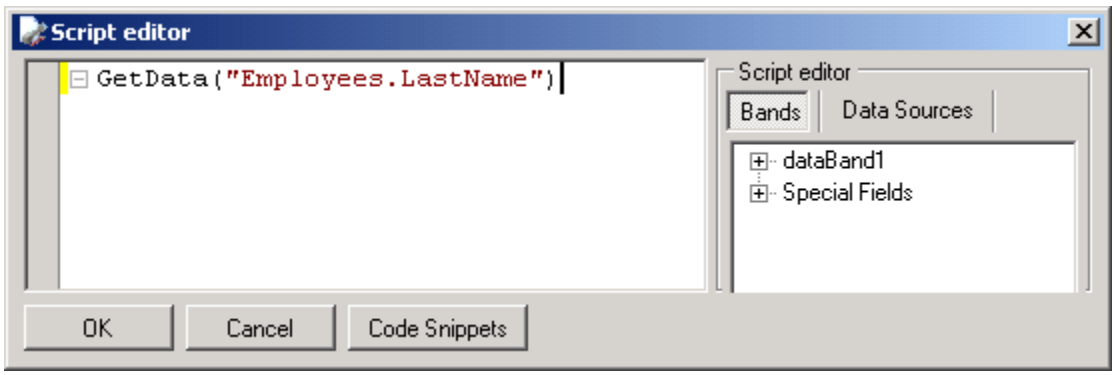


To display the data, add four corresponding *TextBox* controls to the *detail1* band. Set the font size to 11 for each of these controls, too; but don’t boldface them.

Select the first TextBox in detail1, and in the Properties Bindings ...

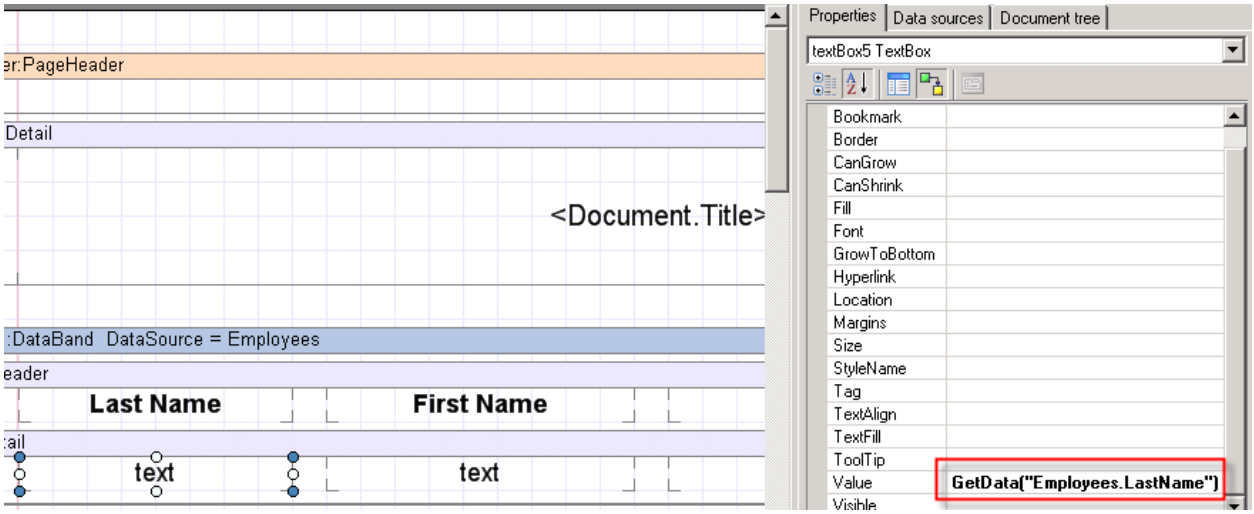


...set a binding to the expression 'GetData("Employees.LastName")' by typing that into the left-side pane of the Script editor[2]:

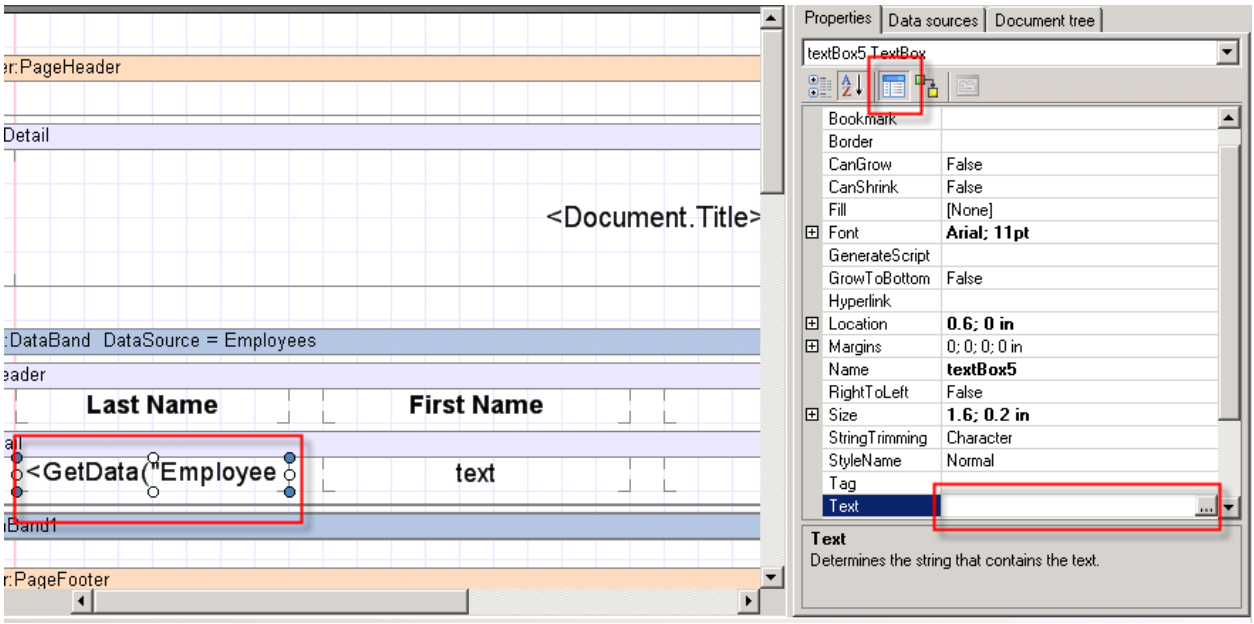


Set values of 'GetData("Employees.FirstName")', 'GetData("Employees.Title")' and 'GetData("Employees.HireDate")' for the other three TextBoxes.

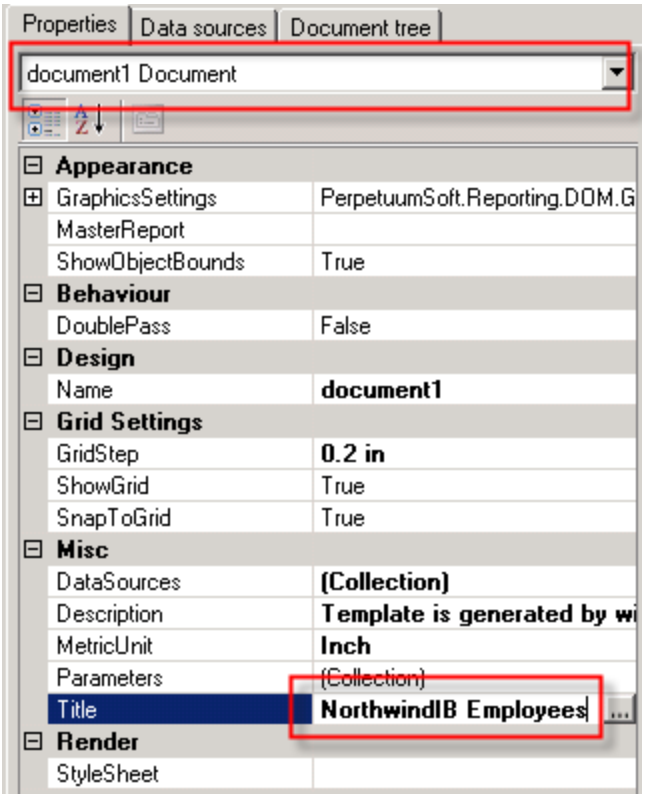
[2] The syntax for the GetData function is as follows: GetData:<DataSourceName>.<PropertyName>



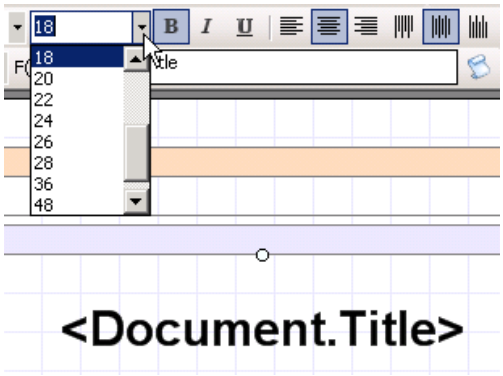
In the toolbar for the Properties tab, click the Properties button to switch from Bindings view to Properties view. Then Clear out the *Text* property of each TextBox in the DetailBand *detail1*, so that the values of the *Value* property that you just set will display in the designer:



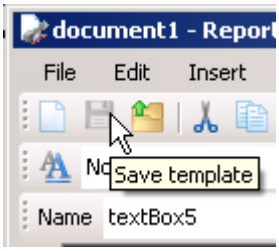
Finally, in the Properties window select the *document1* Document, and set the Title property to NorthwindIB Employees, as shown at right.



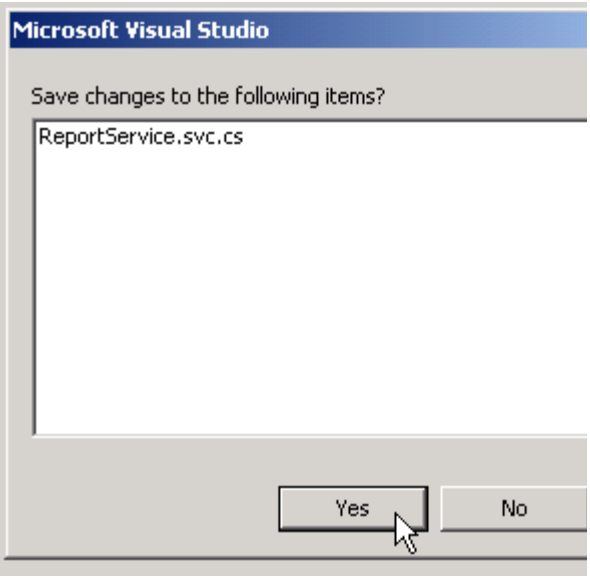
Then in the designer, select the *titleText* TextBox and make it 18-point bold.



Save the completed report template. Close the Report Designer, click <OK> to close the ReportManager editor dialog, and close the ReportService designer.



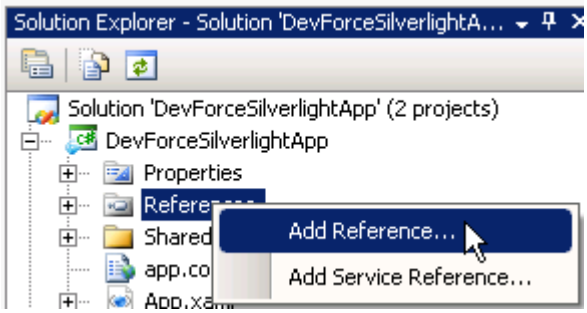
Be sure to indicate “Yes” when the ReportService designer prompts you about whether to save changes you’ve made to the ReportService!

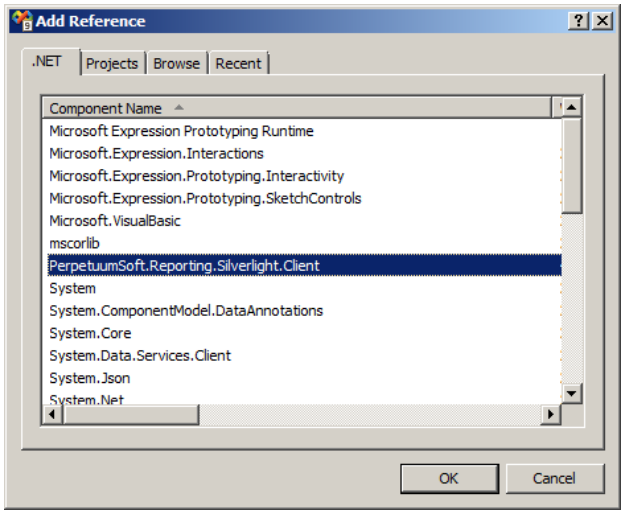


Step 4. Set Up the Client-Side Silverlight ReportViewer Component

To display a report, you must add a report viewer component to the Silverlight application. To do that, first add a reference to the *PerpetuumSoft.Reporting.Silverlight.Client* assembly, which contains the *ReportViewer* component.

To add this reference, right-click the References of the *DevForceSilverlightApp* in the Solution Explorer, choose the Add Reference item from the pop-up menu, and select the *PerpetuumSoft.Reporting.Silverlight.Client* assembly.





In the Silverlight project, create a new *UserControl* name “ReportPage”. Open the *ReportPage.xaml* file and replace it with the following XAML:

XML

```
<UserControl x:Class="DevForceSilverlightApp.ReportPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:rss="clr-
namespace:PerpetuumSoft.Reporting.Silverlight.Client;assembly=Perpetuum
Soft.Reporting.Silverlight.Client"
    Width="Auto"  Height="Auto">
  <Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
      <RowDefinition Height=".15*"/>
      <RowDefinition Height=".85*"/>
    </Grid.RowDefinitions>
    <Grid x:Name="_reportControlsGrid" Grid.Row="0">
      <Grid.RowDefinitions>
        <RowDefinition Height=".5*"/>
        <RowDefinition Height=".5*"/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*"/>
        <ColumnDefinition Width=".25*"/>
        <ColumnDefinition Width=".35*"/>
        <ColumnDefinition Width=".15*"/>
      </Grid.ColumnDefinitions>
      <StackPanel Name="_controls01StackPanel" Orientation="Vertical"
                  Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
                  Margin="10,5,10,5">
        <TextBlock VerticalAlignment="Center"
                   HorizontalAlignment="Left"
                   Margin="0,0,0,0" FontSize="22" FontStyle="Italic">
          FourSimpleSteps</TextBlock>
        <TextBlock VerticalAlignment="Center"
                   HorizontalAlignment="Left"
                   Margin="0,0,0,0" FontSize="22" FontStyle="Italic">
          Reports</TextBlock>
      </StackPanel>
      <TextBlock Grid.Row="0" Grid.Column="1" Grid.RowSpan="2"
                 VerticalAlignment="Center" HorizontalAlignment="Right"
                 Margin="10,0,20,0" FontSize="14">
                 Select Target:</TextBlock>
      <StackPanel Name="_controls02StackPanel" Orientation="Horizontal"
          Grid.Row="0" Grid.Column="2">
        <TextBlock Margin="10,0,10,0" VerticalAlignment="Center">
          Customers:
        </TextBlock>
        <RadioButton Name="_customersRadioButton" IsChecked="false"
                     Margin="10,0,10,0" VerticalAlignment="Center"
                     Checked="_customersRadioButton_Checked">
        </RadioButton>
        <TextBlock Margin="10,0,10,0" VerticalAlignment="Center">
          Limit by Country:
        </TextBlock>
```

```
        <ComboBox Name="_selectCountryComboBox" Margin="10,0,10,0"
                  VerticalAlignment="Center" MinWidth="100"
                  ItemsSource=""
                  ></ComboBox>
      </StackPanel>
      <StackPanel Name="_controls03StackPanel" Orientation="Horizontal"
                  Grid.Row="1" Grid.Column="2">
        <TextBlock Margin="10,0,10,0" VerticalAlignment="Center">
          Employees:</TextBlock>
        <RadioButton Name="_employeesRadioButton" IsChecked="false"
                     Margin="10,0,10,0" VerticalAlignment="Center"

Checked="_employeesRadioButton_Checked"></RadioButton>
      </StackPanel>
      <Button Name="_launchReportButton" Grid.Row="0" Grid.RowSpan="2"
              Grid.Column="3" Click="_launchReportButton_Click"
              HorizontalAlignment="Left" MinWidth="100"
Margin="10,10,10,10">
        <TextBlock FontSize="16">Go</TextBlock>
      </Button>
    </Grid>
    <rss:ReportViewer x:Name="_reportViewer" Grid.Row="1"/>
  </Grid>
</UserControl>
```

Note the inclusion of the *PerpetuumSoft* namespace as an attribute of the ReportPage UserControl element, and the insertion of the *ReportViewer* component in Row 1 of the outer grid:

XML

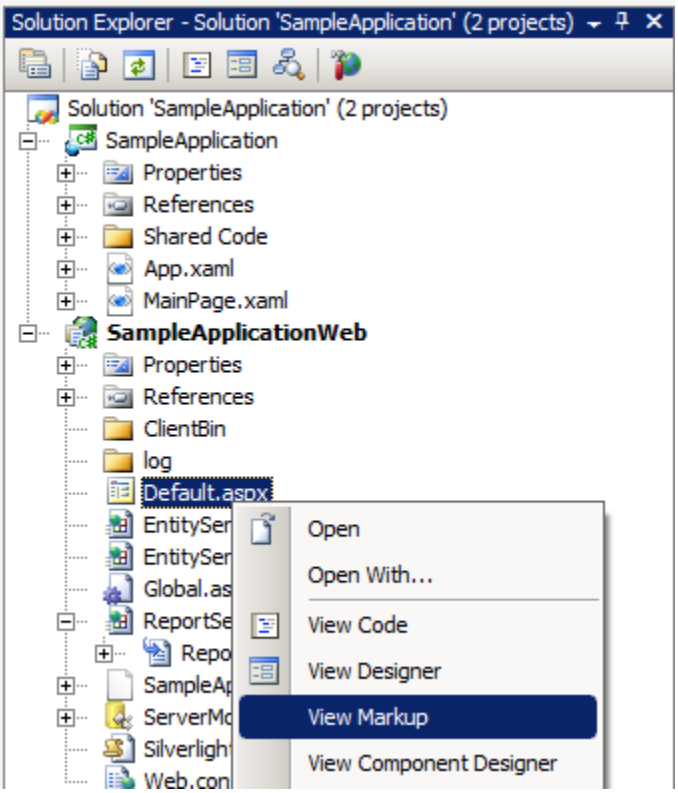
```
Xmlns:rss="clr-
namespace:PerpetuumSoft.Reporting.Silverlight.Client;assembly=Perpetuum
Soft.Reporting.Silverlight.Client"

...

<rss:ReportViewer x:Name="_reportViewer" Grid.Row="1"/>
```

Note also that we have set the value of the Width and Height attributes of the UserControl to "Auto", so it will fill the browser screen and give the ReportViewer as much real estate as possible.

Save and close the ReportPage.xaml file.

For the Report Viewer to do its job, it must be able to find the URL address of the report service. Let's store that in the aspx page that will contain the ReportPage UserControl.

View the MarkUp code for the *Default.aspx* page:



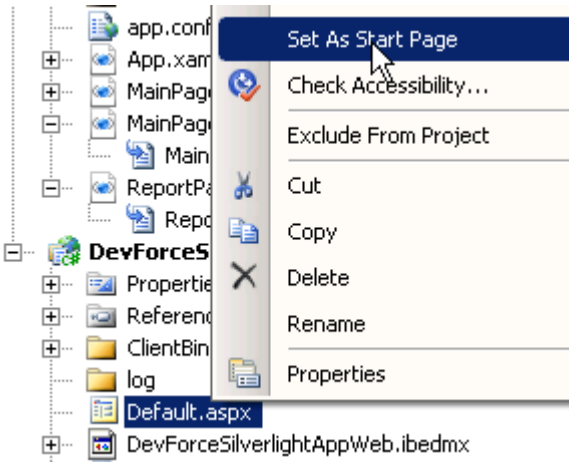
Within the <body> section of the markup code, find the <object> element, and add the "initParams" parameter as shown below:

XML

```
<object data="data:application/x-silverlight-2," type="application/x-silverlight-2" width="100%" height="100%">
  <param name="source" value="ClientBin/SampleApplication.xap"/>
  <param name="onError" value="onSilverlightError" />
  <param name="background" value="white" />
  <param name="minRuntimeVersion" value="3.0.40624.0" />
  <param name="autoUpgrade" value="true" />
  <param name="initParams" value='<%= string.Format("http://{0}{1}", Request.Url.Authority, ResolveUrl("~/ReportService.svc")) %>' />
  <a href="http://go.microsoft.com/fwlink/?LinkID=149156&v=3.0.40624.0" style="text-decoration:none">
  <img src="http://go.microsoft.com/fwlink/?LinkId=108181" alt="Get Microsoft Silverlight" style="border-style:none"/>
  </a>
</object>
```

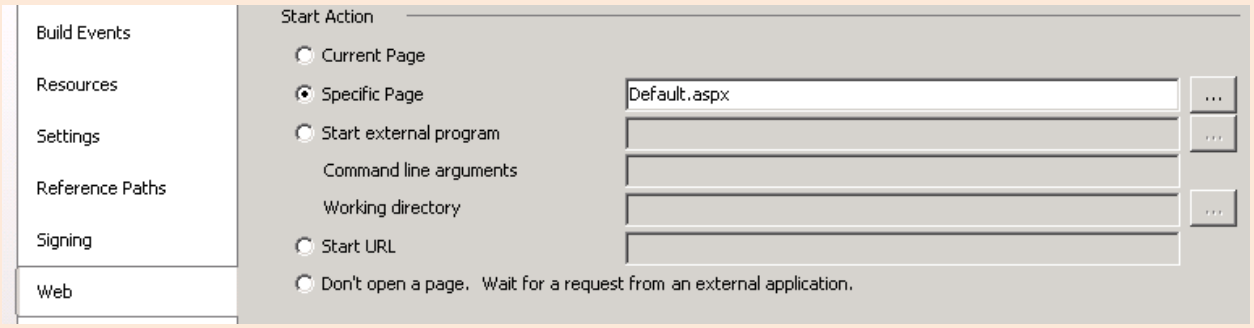
Close *default.aspx*, saving your work.

We'll take this opportunity to remind you to make sure that, as instructed earlier in this walk-through, you've made *default.aspx* the start page for the web project (and therefore the app). Right-click it, and select Set As Start Page from the shortcut menu:



This is important because the default start page for web apps is the "Current Page" -- and that can get changed simply by opening a different page in the web project for editing work. Explicitly setting *default.aspx* to the StartPage will mean our app will always start where we mean for it to start!

Note: You can always see what the web project's start page is by right-clicking its node in the Solution Explorer, selecting Properties, and then viewing the Web tab of the Properties dialog:



Now return to the *ReportPage* UserControl and view its "code behind". Replace that with the following code:

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
```

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Collections.ObjectModel;

using DomainModel;
using IdeaBlade.EntityModel;

namespace DevForceSilverlightApp {
  public partial class ReportPage : UserControl {
    public ReportPage() {
      InitializeComponent();
      _reportViewer.ServiceUrl =
        System.Windows.Browser.HtmlPage.Plugin
        .GetProperty("initParams").ToString();
      this.Loaded += new RoutedEventHandler(ReportPage_Loaded);
    }

    void ReportPage_Loaded(object sender, RoutedEventArgs e) {
      IEntityQuery<string> countryNames =
        _mgr.Customers.Select(c => c.Country).Distinct()
          .OrderBy(s => s);
      _mgr.ExecuteQueryAsync<string>(countryNames,
         GotCountryNames, null);
      InitRadioButtons();
    }

    /// <summary>
    /// Inits the Checked state of the RadioButtons
    /// </summary>
    /// <remarks>
    /// First tried to do this in XAML, but this made the form blow up
for
    /// RadioButtons that had a Checked handler and had IsChecked="true"
    /// in the XAML.
    /// </remarks>
    private void InitRadioButtons() {
      _customersRadioButton.IsChecked = false;
      _employeesRadioButton.IsChecked = true;
    }

    private void GotCountryNames(EntityFetchedEventArgs<string> args) {
      ObservableCollection<string> countries =
        new ObservableCollection<string>();
      countries.Add(_selectAllToken);
      foreach (string countryName in args.Result) {
        countries.Add(countryName);
      }
      _selectCountryComboBox.ItemsSource = countries;
    }

    private void _launchReportButton_Click(
        object sender, RoutedEventArgs e) {

      if ((bool)_employeesRadioButton.IsChecked) {
        // run EmployeesReport
```

```
        _reportViewer.ReportName = "EmployeesReport";
      }
      else {
        // run CustomersReport
        _reportViewer.ReportName = "CustomersReport";

        string filterCountry =
         (string)_selectCountryComboBox.SelectedItem;
        if (filterCountry == _selectAllToken) {
          filterCountry = null;
        }

        if (_reportViewer.Parameters.Keys.Contains("Country")) {
          _reportViewer.Parameters["Country"] = filterCountry;
        }
        else {
          _reportViewer.Parameters.Add("Country", filterCountry);
        }
      }

      _reportViewer.RenderDocument();
    }

    private void _customersRadioButton_Checked(
        object sender, RoutedEventArgs e) {
      _employeesRadioButton.IsChecked = false;
    }

    private void _employeesRadioButton_Checked(
        object sender, RoutedEventArgs e) {
      _customersRadioButton.IsChecked = false;
    }

    #region Private Fields
    DomainModelEntityManager _mgr =
      DomainModelEntityManager.DefaultManager;
    const string _selectAllToken = "<All>";
    #endregion Private Fields

  }
}
```

Note how the ReportPage constructor calls the html page's *Plugin.GetProperty()* method to find the URL of the report service. The Click handler for the *_launchReportButton* (`_launchReportButton_Click`) checks the RadioButtons to determine which report was selected. If the user selected the CustomersReport[3], finds what Country the user selected and

[3] Remember: for brevity we did not create the CustomersReport in this tutorial; but it is present in the Visual Studio solution that accompanies this article. The code in the solution you've created here is ready for that report as soon as it gets created.

adds that to the *ReportViewer*'s *Parameters* collection so it will be available to the server-side processing logic you previously added to the *ReportService.svc.cs* file.

Invoking *RenderDocument()* causes the report to be rendered on the server and returned to the Report Viewer for display.

ReportPage is now ready to go, but we need a way to display it from *MainPage*. Open the markup for *MainPage* and add a button. Add the XAML for this button near the end of the file, right after the closing element for the *ScrollViewer* and before the closing element for the main *Grid*, as shown:

XML

```
</ScrollViewer>
...
<Button x:Name="reportsButton" Content="Reports"
  Click="reportsButton_Click"
  Width="80" Height="50" Grid.Row="1" Grid.Column="0">
</Button>
...
</Grid>
```

Now add a Click event handler for the *reportsButton* to the MainPage code:

C#

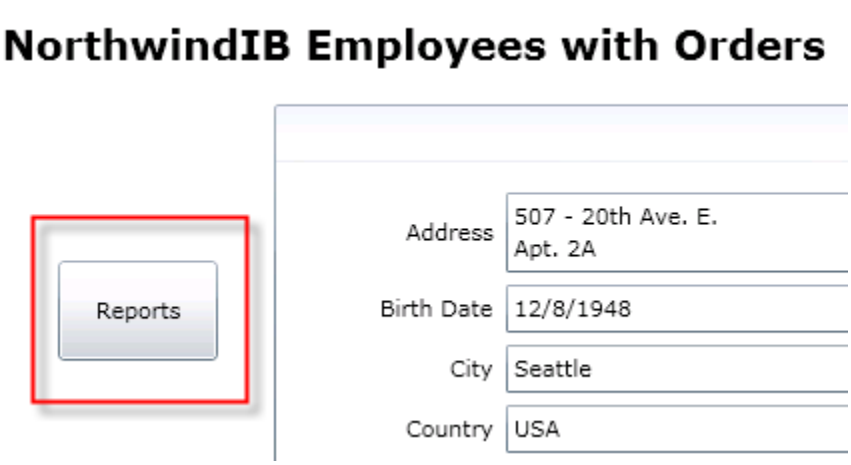
```
private void reportsButton_Click(object sender, RoutedEventArgs e) {
  this.Content = new ReportPage();
}
```

Save your work and close the *MainPage* file.

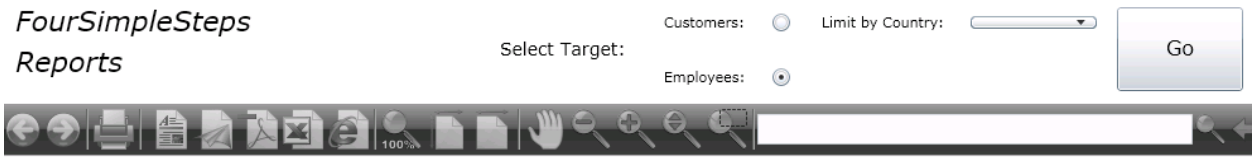
Step 5. Launch the Application

Launch the application by clicking the Start Debugging button on the main Visual Studio toolbar.

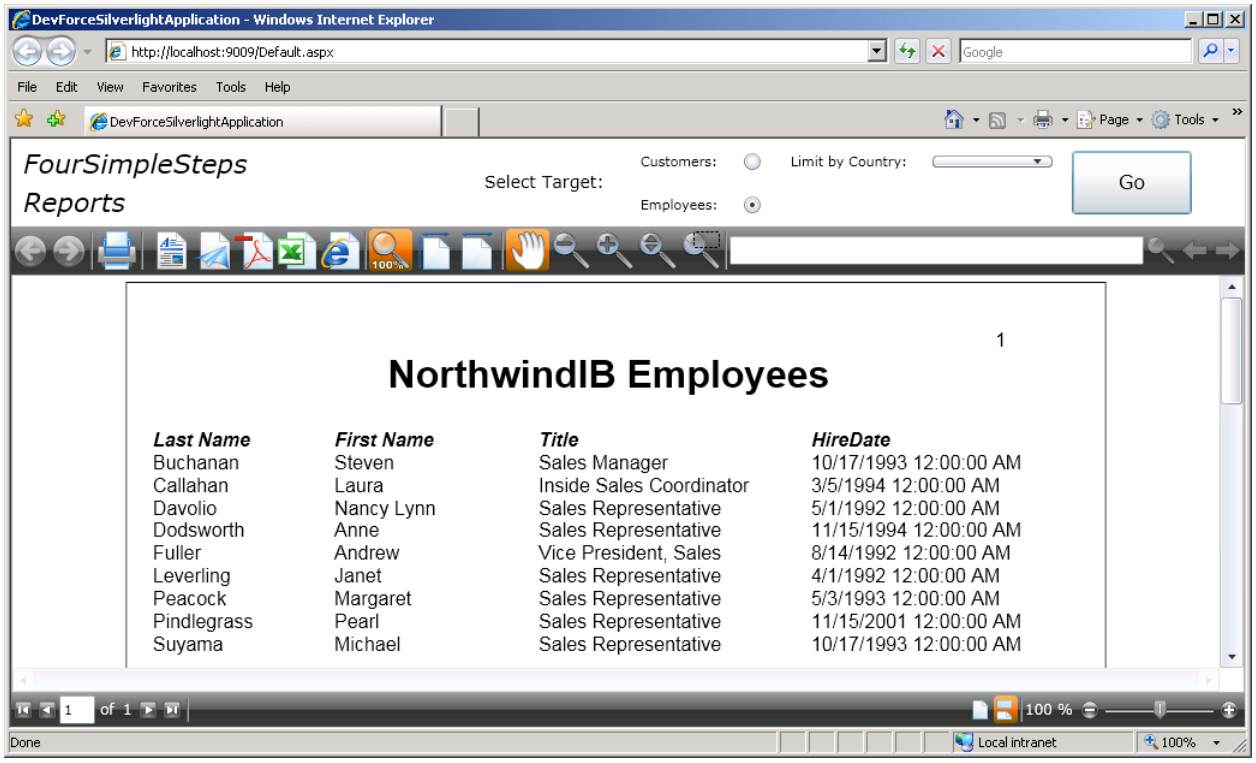
On the MainPage, click the ReportButton. (You don't actually have to wait for the data to get loaded into MainPage.)



With the controls at the top of the ReportPage, select Employees as the reporting target and click the *Go* button:

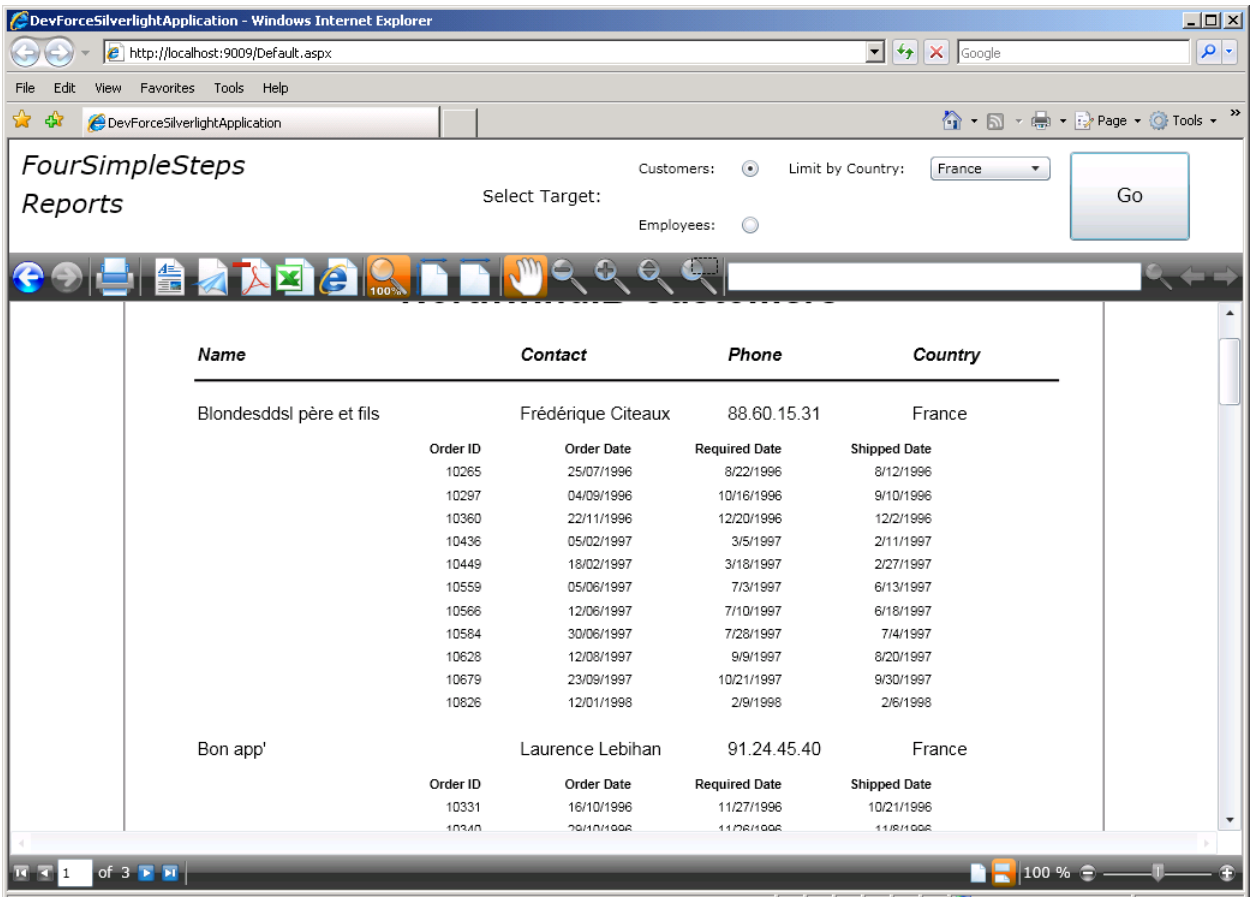


You should see something like this:

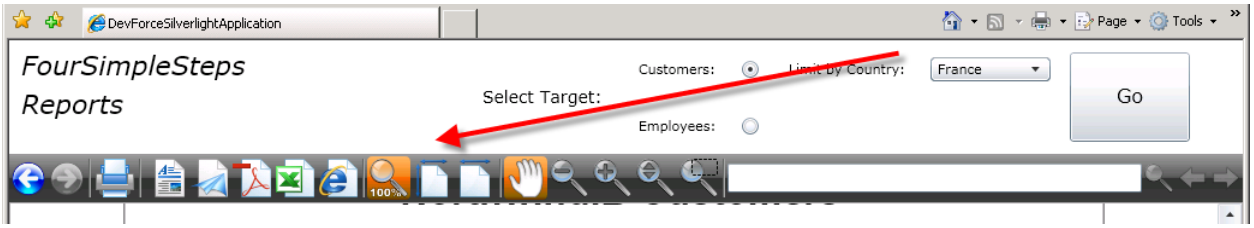


In the code solution that accompanies this article, we've added the CustomersReport, configuring it to display both Customers and their associated Orders. The client- and server-side logic you already added makes the filter (by Country) operational.

Here's what the CustomersReport looks like, filtered to display Customer headquartered in France:



Note the buttons on the toolbar that's built-in to the ReportSharpShooter *ReportViewer* component.



You can use them to do all of the following operations with the displayed report:

1. Print it to any connected printer;
2. Export it as an RTF, XPS, PDF, Microsoft Excel, or HTML document;
3. Zoom and pan it;
4. Search it.

We get all of those capabilities with no additional programmer whatsoever. Try them out!

Conclusion

You've walked through all of the steps necessary, using the PerpetuumSoft *ReportSharpShooter* tool, to add reporting capability to the *Four Simple Steps* application. You can obviously design far more sophisticated reports and report logic than you've seen here, but this should get you going. See the *ReportSharpShooter* documentation for detail about exploiting the capabilities of this powerful Silverlight reporting tool!

Appendix A
Completed *Web.config* File Contents

XML

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="ideablade.configuration"
type="IdeaBlade.Core.Configuration.IdeaBladeSection, IdeaBlade.Core" />
  </configSections>
  <ideablade.configuration version="5.00" updateFromDomainModelConfig="Ask"
clientApplicationType="Silverlight">
    <logging logFile="log\DebugLog.xml" />
    <edmKeys>
      <edmKey name="Default"
connection="metadata=res://DevForceSilverlightApp/ServerModelNorthwindIB.csdl|res://DevForce
SilverlightApp/ServerModelNorthwindIB.ssdl|res://DevForceSilverlightApp/ServerModelNorthwind
IB.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=.;Initial
Catalog=NorthwindIB;Integrated Security=True;MultipleActiveResultSets=True&quot;"
containerName="DevForceSilverlightApp.ServerModelNorthwindIBContext">
        <probeAssemblyNames>
          <probeAssemblyName name="DevForceSilverlightApp" />
        </probeAssemblyNames>
      </edmKey>
    </edmKeys>
  </ideablade.configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="webBehavior">
          <webHttp/>
        </behavior>
      </endpointBehaviors>
      <serviceBehaviors>
        <behavior name="DevForceSilverlightApp.ReportingServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
        <behavior name="DevForceSilverlightApp.ReportServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
    <services>
      <service name="EntityService">
        <endpoint address="" binding="customBinding"
bindingConfiguration="customBinaryBinding"
          contract="IdeaBlade.EntityModel.IEntityServiceContract" />
      </service>
      <service name="IdeaBlade.EntityModel.Server.EntityServer">
        <endpoint address="" binding="customBinding"
bindingConfiguration="customBinaryBinding"
          contract="IdeaBlade.EntityModel.IEntityServerContract" />
      </service>

      <service behaviorConfiguration="DevForceSilverlightApp.ReportServiceBehavior"
        name="DevForceSilverlightApp.ReportService">
        <endpoint address="" binding="basicHttpBinding"
           bindingConfiguration=""
           contract="PerpetuumSoft.Reporting.Silverlight.Server.IReportService" />
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
        <endpoint address="rest" behaviorConfiguration="webBehavior"
            binding="webHttpBinding"
            contract="PerpetuumSoft.Reporting.Silverlight.Server.IReportServiceResources"/>
```

```
      </service>

    </services>
    <bindings>
      <customBinding>
        <binding name="customBinaryBinding">
          <binaryMessageEncoding>
            <readerQuotas maxDepth="2147483647" maxStringContentLength="2147483647"
              maxArrayLength="2147483647" />
          </binaryMessageEncoding>
          <httpTransport maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
        </binding>
        <binding name="customBinding0">
          <binaryMessageEncoding />
          <httpTransport />
        </binding>
        <binding name="customBinding1">
          <binaryMessageEncoding />
          <httpTransport />
        </binding>
      </customBinding>
    </bindings>
  </system.serviceModel>
  <system.web>
    <!--
            Set compilation debug="true" to insert debugging
            symbols into the compiled page. Because this
            affects performance, set this value to true only
            during development.
        -->
    <compilation debug="true">
      <assemblies>
        <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089" />
        <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089" />
        <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" />
        <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089" />
        <add assembly="System.Data.Entity, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" />
      </assemblies>
    </compilation>
    <pages>
      <controls>
        <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
        <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" />
      </controls>
    </pages>
    <httpHandlers>
      <remove verb="*" path="*.asmx" />
      <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
      <add verb="*" path="* AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
      <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35" validate="false" />
    </httpHandlers>
    <httpModules>
      <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
    </httpModules>
  </system.web>
  <system.codedom>
```

```
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">
        <providerOption name="CompilerVersion" value="v3.5" />
        <providerOption name="WarnAsError" value="false" />
      </compiler>
    </compilers>
  </system.codedom>
  <!--  The system.webServer section is required when hosting a Silverlight application
under Internet
        Information Services 7.0.  It is not necessary for previous version of IIS.
    -->
  <system.webServer>
    <validation validateIntegratedModeConfiguration="false" />
    <modules>
      <remove name="ScriptModule" />
      <add name="ScriptModule" preCondition="managedHandler"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
    </modules>
    <handlers>
      <remove name="WebServiceHandlerFactory-Integrated" />
      <remove name="ScriptHandlerFactory" />
      <remove name="ScriptHandlerFactoryAppServices" />
      <remove name="ScriptResource" />
      <add name="ScriptHandlerFactory" verb="*" path="*.asmx" preCondition="integratedMode"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
      <add name="ScriptHandlerFactoryAppServices" verb="*" path="*_AppService.axd"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
      <add name="ScriptResource" preCondition="integratedMode" verb="GET,HEAD"
path="ScriptResource.axd" type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
    </handlers>
  </system.webServer>
  <connectionStrings>
    <add name="ServerModelNorthwindIBContext"
connectionString="metadata=res://*/ServerModelNorthwindIB.csdl|res://*/ServerModelNorthwindI
B.ssdl|res://*/ServerModelNorthwindIB.msl;provider=System.Data.SqlClient;provider connection
string=&amp;quot;Data Source=.;Initial Catalog=NorthwindIB;Integrated
Security=True;MultipleActiveResultSets=True&amp;quot;" providerName="System.Data.EntityClient"
/>
  </connectionStrings>
</configuration>
```