SharpShooter Reports.Web
Getting Started for WCF

Last modified on: September 5, 2011

Table of Content

System Requirements

In order to use SharpShooter Reports.Web successfully in Web applications you will need:

- .NET Framework 3.5 or higher
- Visual Studio 2008/2010

Introduction

The purpose of this getting started is to cover the basic use and provide minimum required skills necessary for work with SharpShooter Reports.Web. This is a step-by-step tutorial for the creation of a web application with SharpShooter Reports.Web. Let's examine the creation and configuration of a service, creation of a report and integration of the report web viewer.

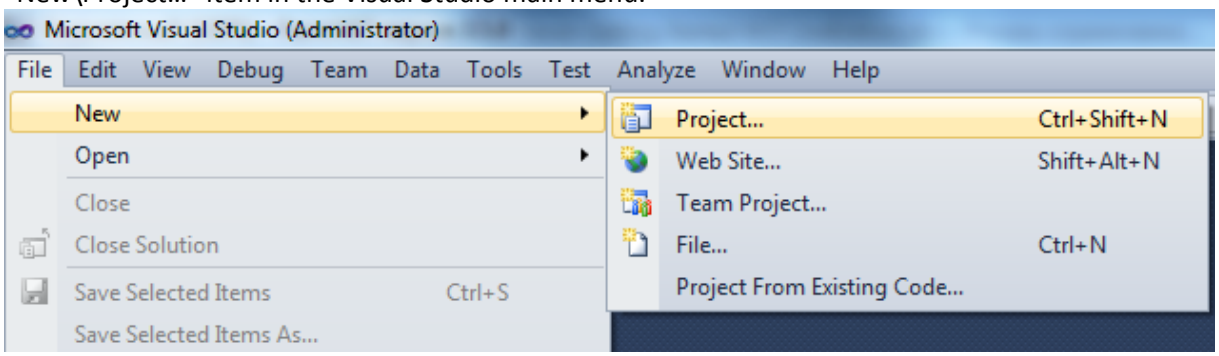
The steps from 1 to 13 describe the creation and setting of a server part of an application.

The steps from 14 to 21 describe the setting of a client part of an application.

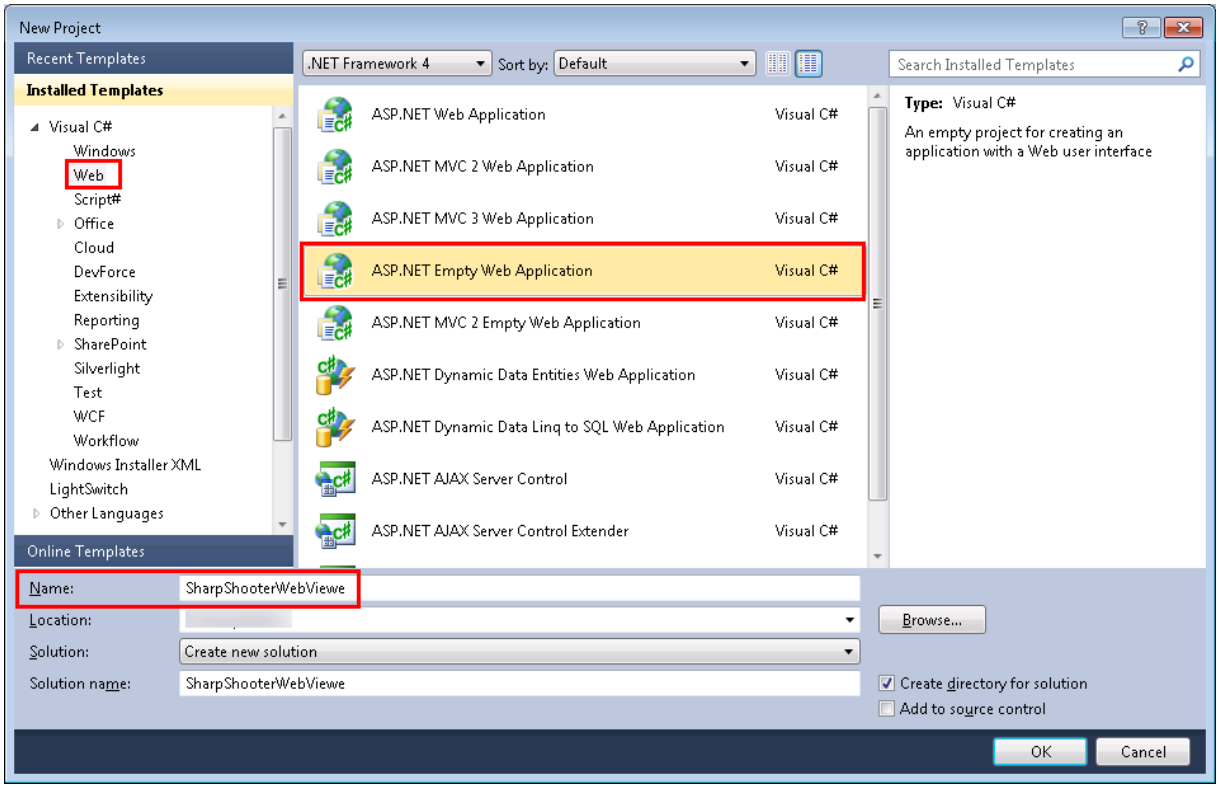
Creation of a Web application

Step 1. Creation of a Web project

Create a new "ASP.NET Web Application" project with the "SharpShooterWebViewer" name. Select the "New\Project…" item in the Visual Studio main menu.

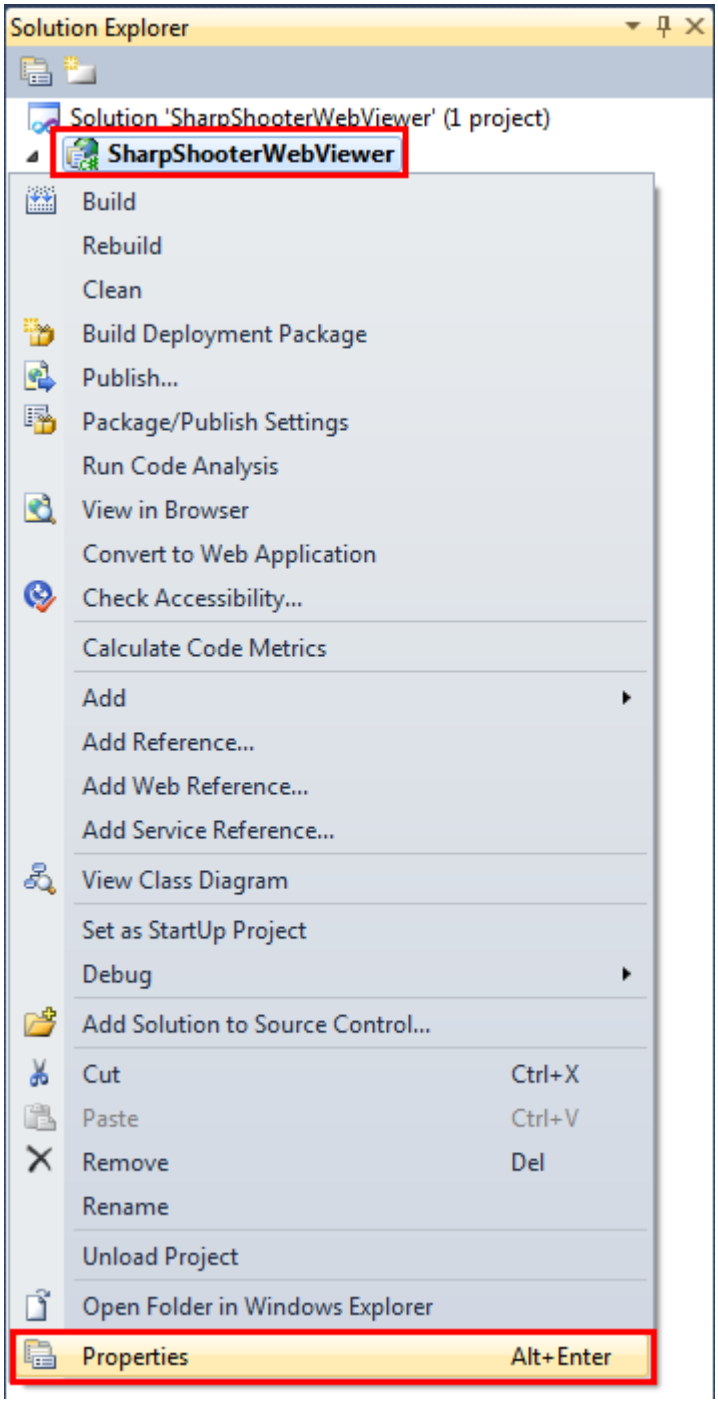


Select "ASP.NET Web Application", enter "SharpShooterWebViewer" as a project name and click the "OK" button.

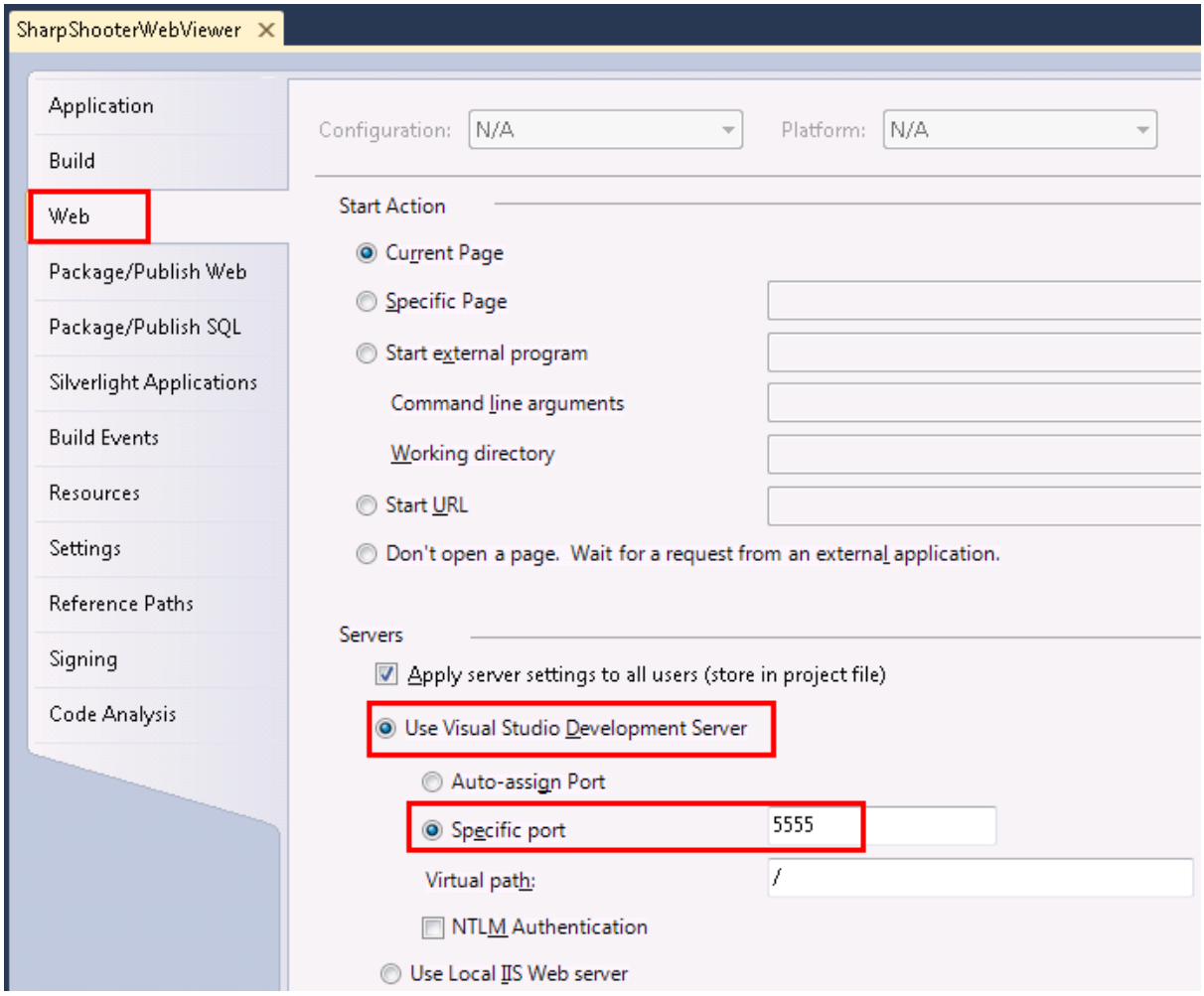


Step 2. Setting up a Web application

Invoke the context menu and select the “Properties” item. Here you can change the properties of the “SharpShooterWebViewer” project.

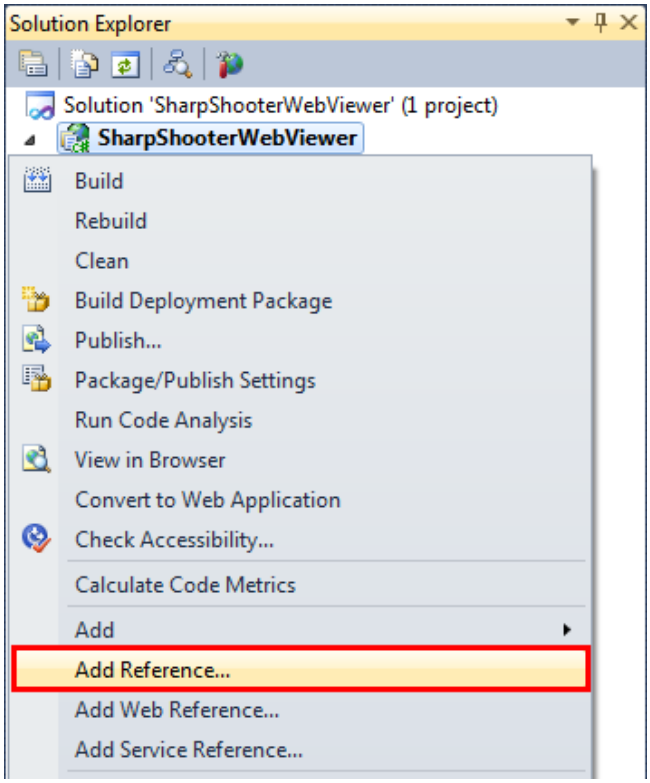


Set “Use Visual Studio Development Server”, “Specific port” equal to 5555 in the “Web” tab.



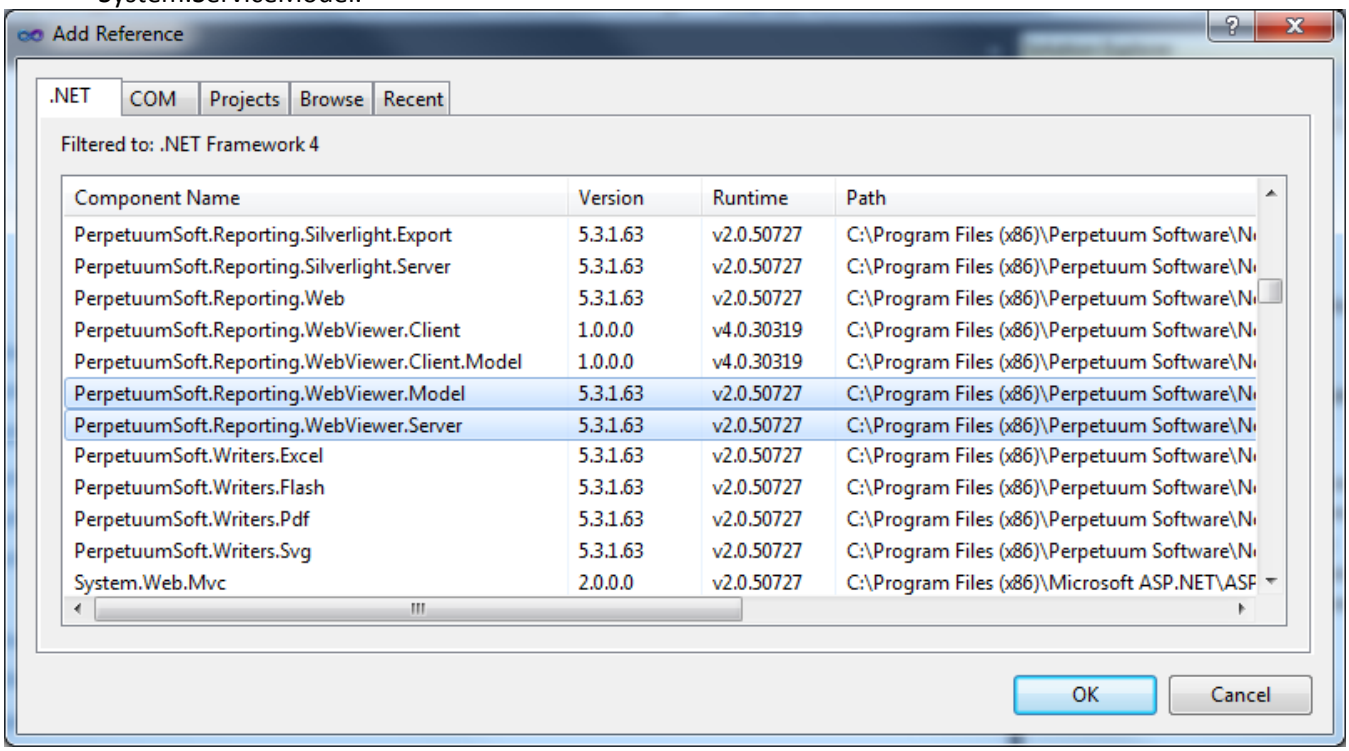
Step 3. Addition of reference to the assemblies

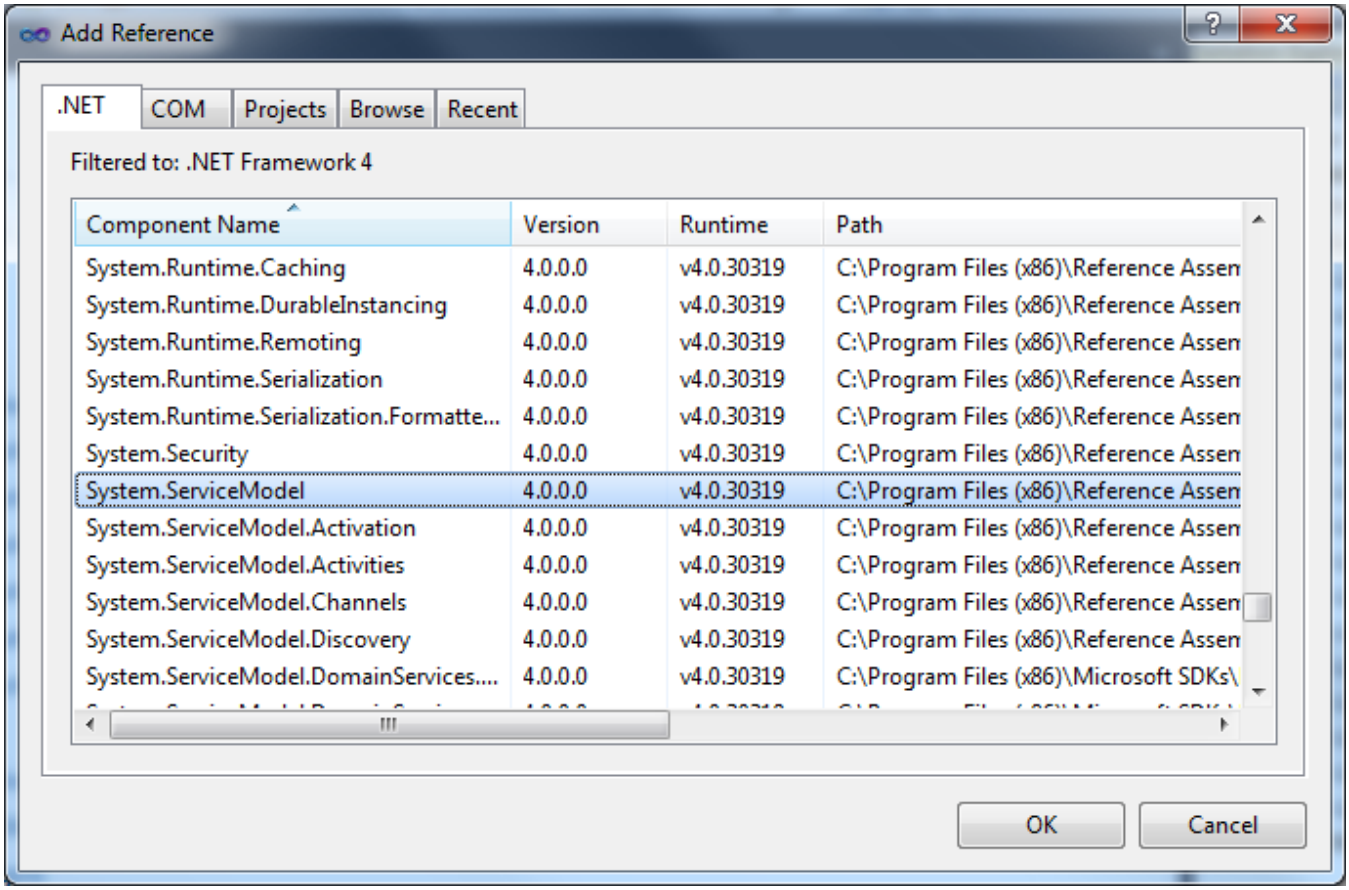
You should add references to the assemblies. Select the “Add Reference” item in the “Solution Explorer” in the context menu of the “SharpShooterWebViewer” project.



Add the following assemblies to the project:

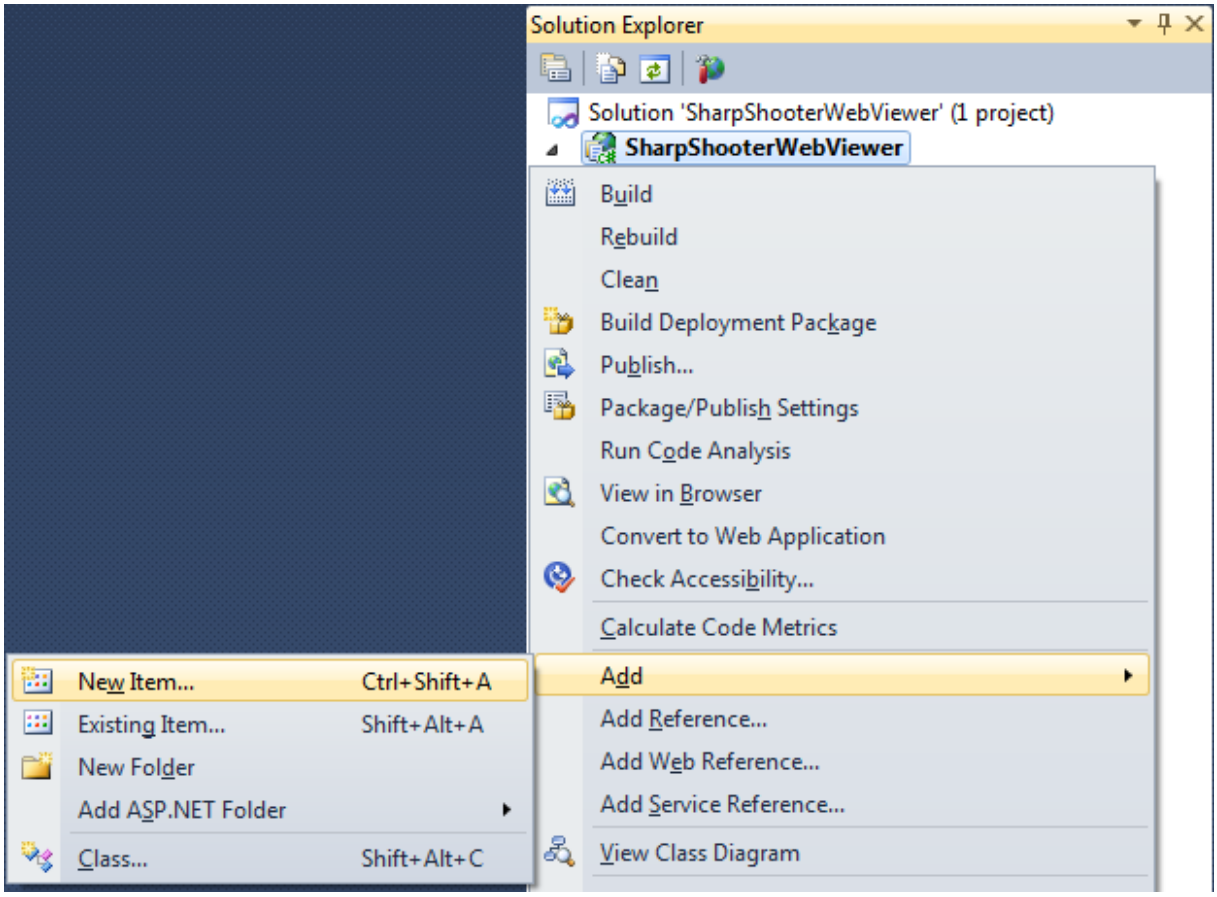
- PerpetuumSoft.Reporting.WebViewer.Model;
- PerpetuumSoft.Reporting.WebViewer.Server;
- System.ServiceModel.



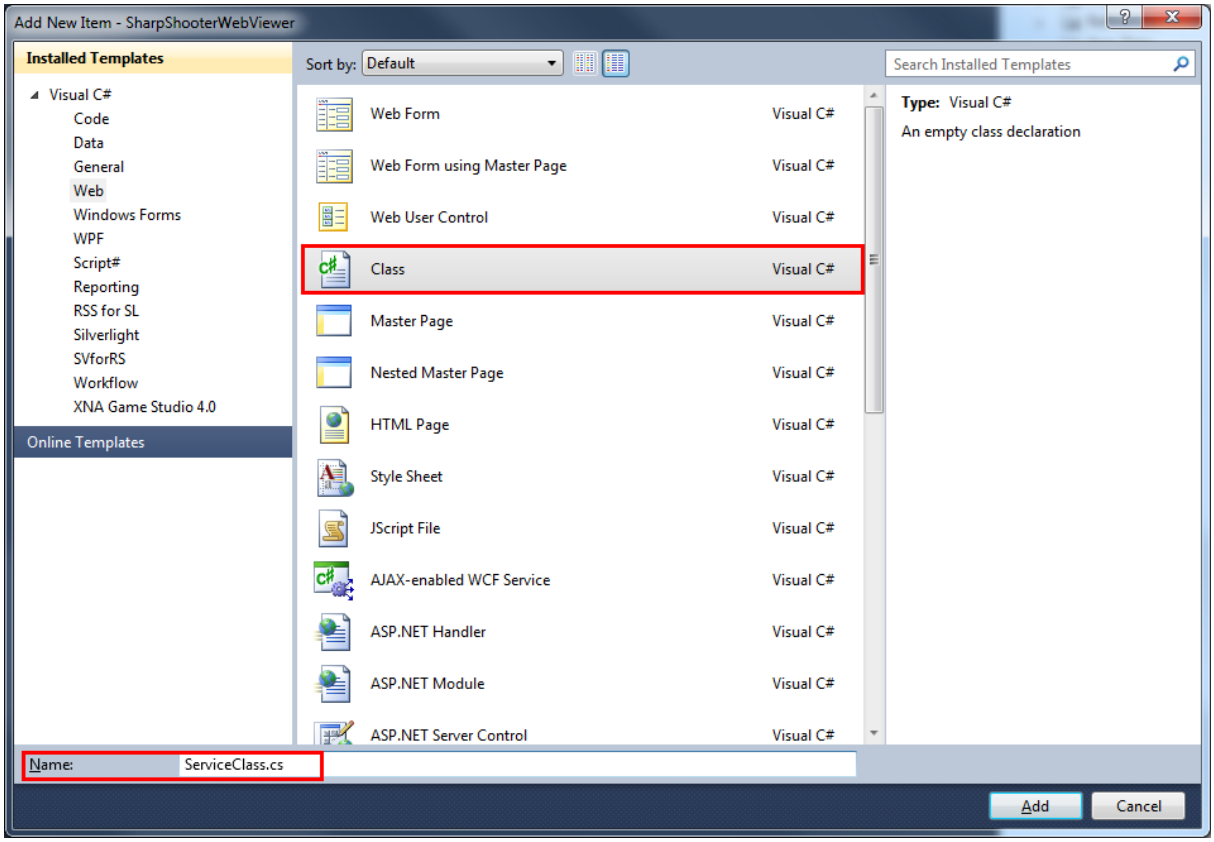


Step 4. Adding report service

Add the class, which implements the logic of the service work, to the project. Invoke the context menu of the “SharpShooterWebViewer” project and select the “Add\New Item…” item.



Select the “Class” element. Set name of the “ServiceClass.cs” class in the “Name” field.



Add the “System.ServiceModel.Activation”, “System.Data” and “PerpetuumSoft.Reporting.WebViewer.Server” namespaces to the “ServiceClass”. To do this, use the using directive:

```
using System.ServiceModel.Activation;
using System.Data;
using PerpetuumSoft.Reporting.WebViewer.Server;
```

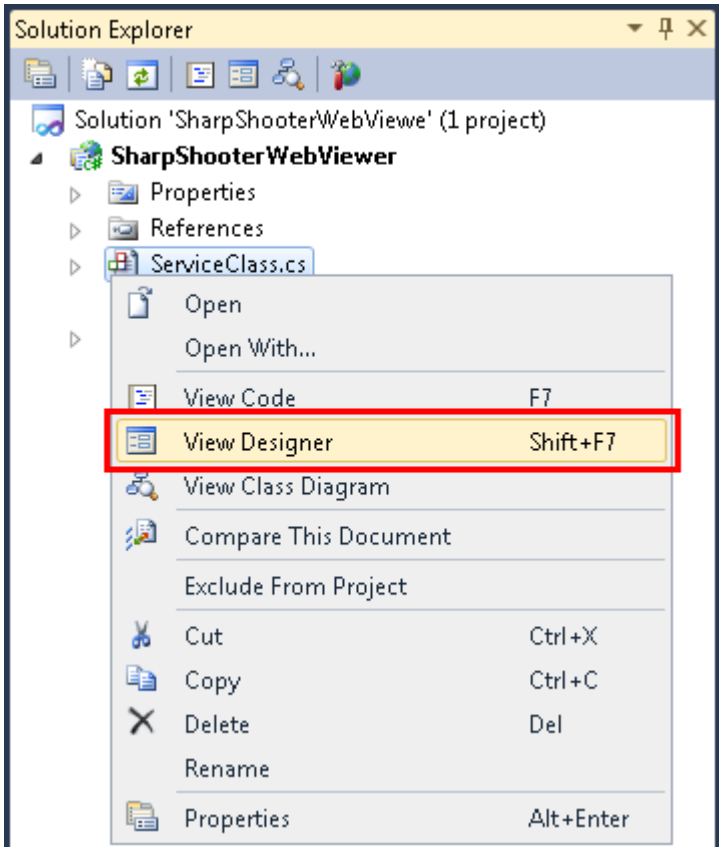
Inherit the added class from the “ReportServiceBase” class. Thus, you get ready implementation and ability to change standard behavior. You should mark the service class with AspNetCompatibilityRequirements attribute. After this the service will have access to ASP.NET context. The service needs the ASP.NET context to cache document’s data if other cache mechanism is not implemented.

Add the “InitializeComponent” function and the constructor, which invokes this function, to the “ServiceClass” class.

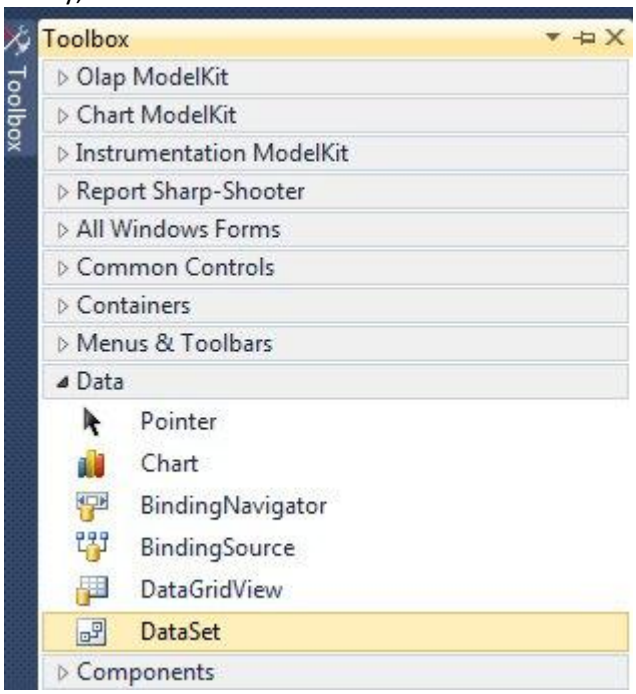
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ServiceModel.Activation;
using System.Data;
using PerpetuumSoft.Reporting.WebViewer.Server;

namespace SharpShooterWebViewer
{
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Required)
]
    public class ServiceClass : ReportServiceBase
    {
        public ServiceClass()
        {
            InitializeComponent();
        }
        private void InitializeComponent()
        { }
    }
}
```

Step 5. Creation of data source

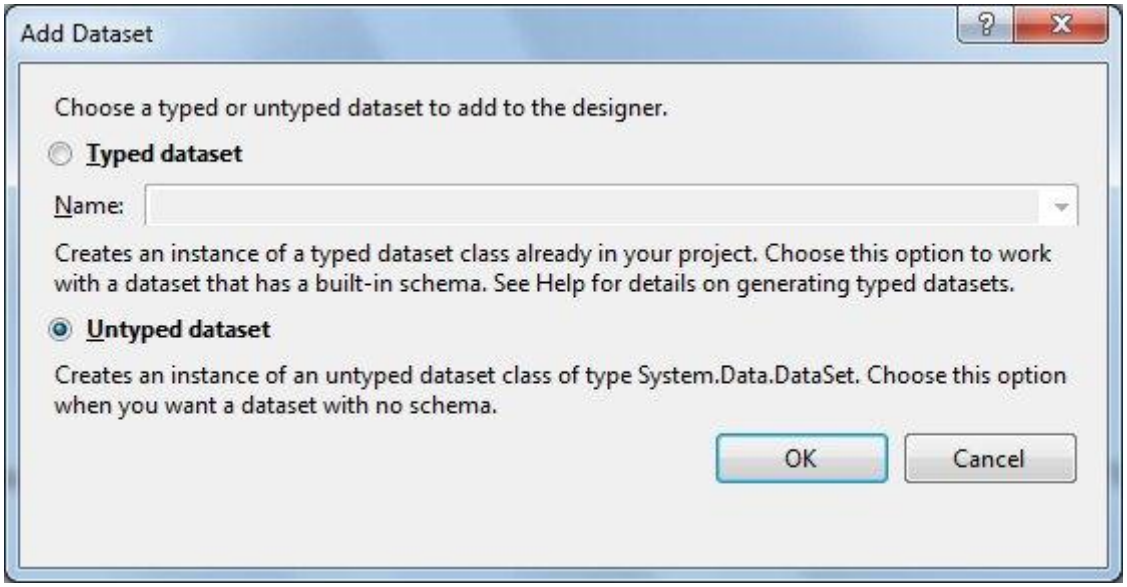
Open the designer for the “ServiceClass”. Select the “View Designer” item in “Solution Explorer” in the context menu of the “ServiceClass.cs” file.



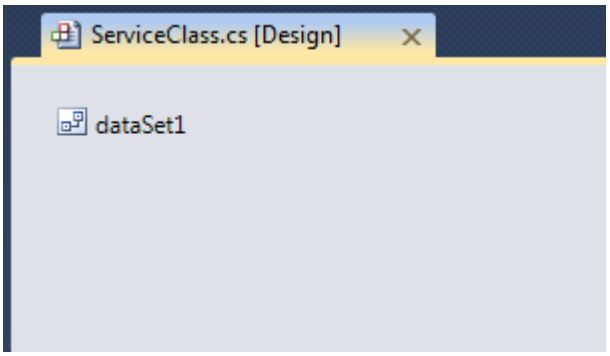
Firstly, create data source structure. Add “Dataset” from the “Toolbox” (double click on the Dataset in Toolbox).



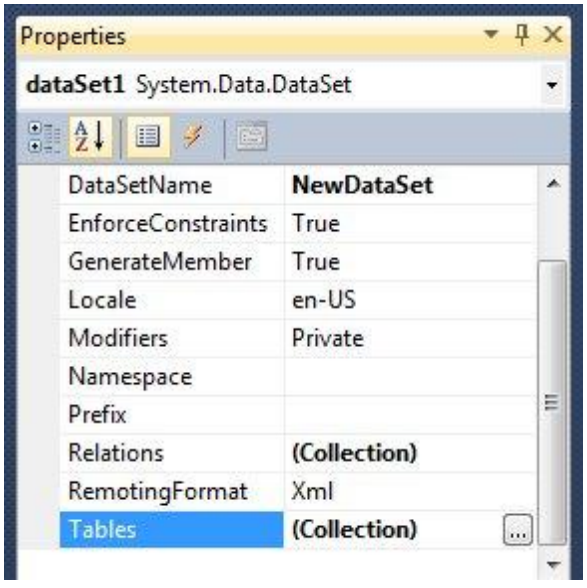
And select the Untyped dataset.



After that the dataset node (dataSet1) will appear in the designer.

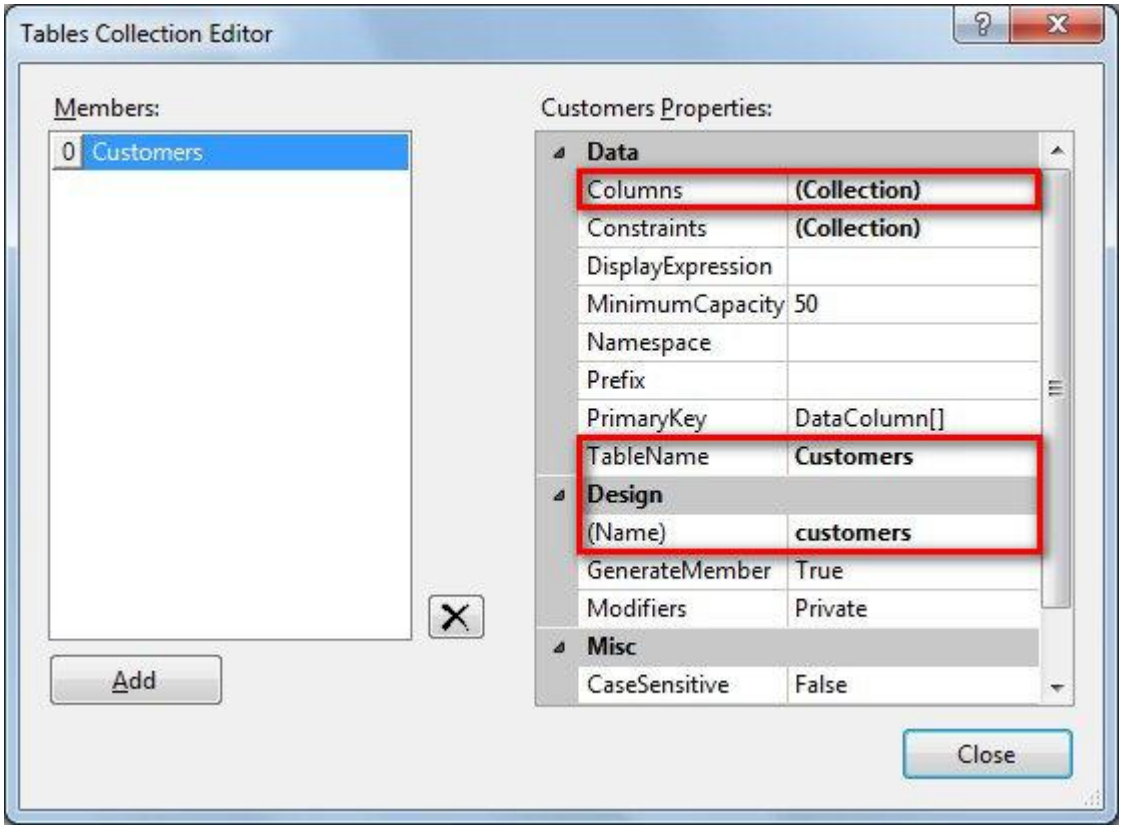


Open Tables Collection Editor (click button  in property Tables).

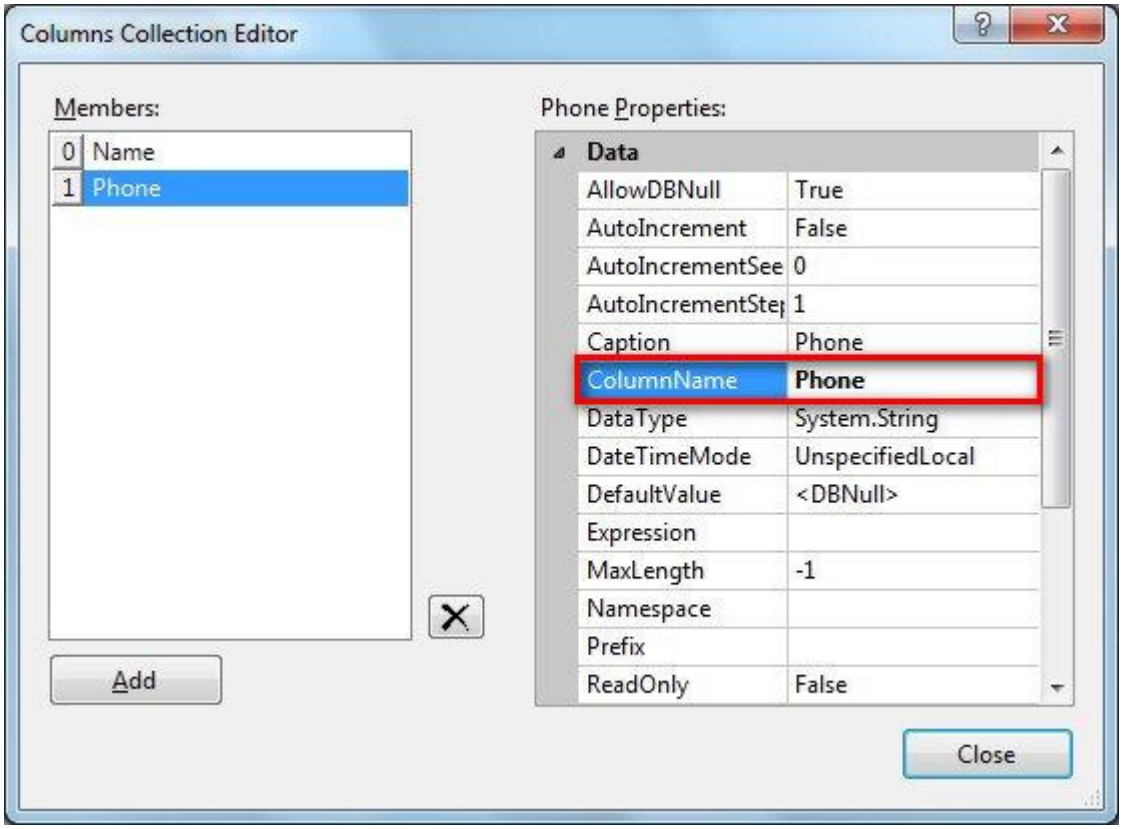


Add Customers table to the dataSet1 (click “Add” button and set TableName property value to “Customers”, set Name property value to “customers”).

After that open Columns Collection Editor (click button in property grid of Tables Collection Editor)

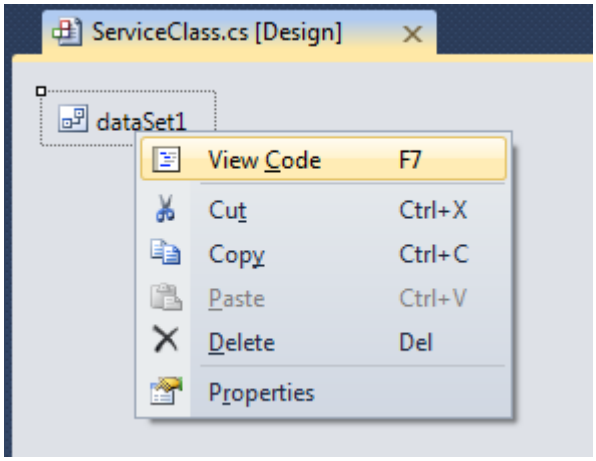


Add two columns by clicking “Add” button and setting ColumnName property value to “Name” and “Phone”.



Step 6. Adding data to service

Data structure is defined, and it’s necessary to fill Customers Table with the data. To view source code right-click on designer area and click “View Code” item in contextual menu.

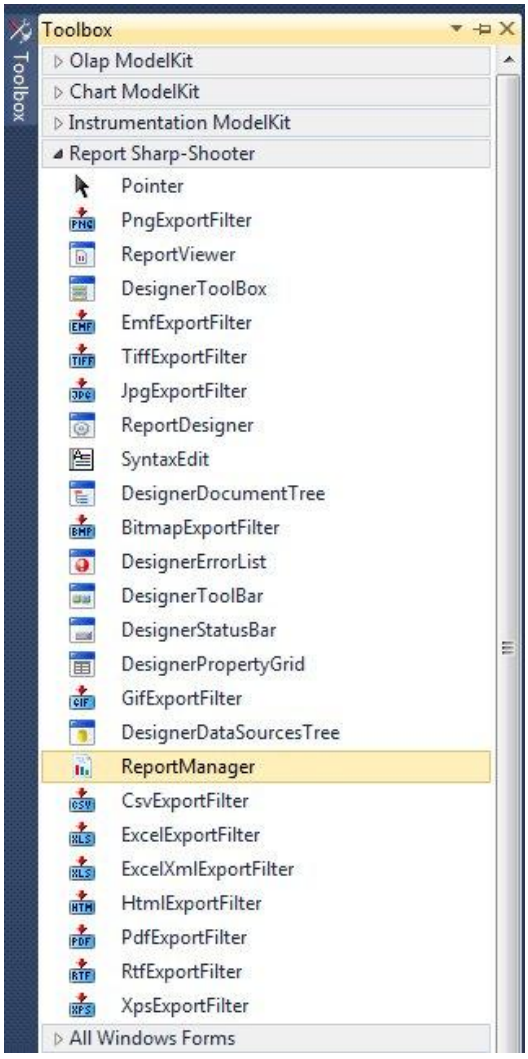


You should fill data sources with the values via overriding OnLoadData method of the ServiceClass class.

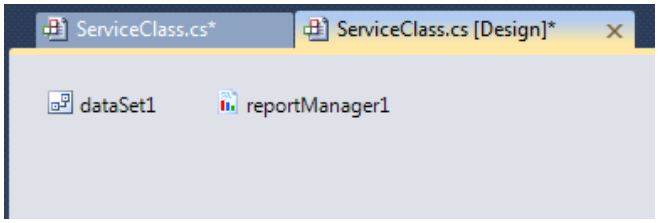
```
protected override void OnLoadData(IDictionary<string, object> parameters, string reportName,
PerpetuumSoft.Reporting.Components.ReportSlot reportSlot)
{
    base.OnLoadData(parameters, reportName, reportSlot);
    DataRow row = customers.NewRow();
    row["Name"] = "Johnson Leslie";
    row["Phone"] = "613-442-7654";
    customers.Rows.Add(row);
    row = customers.NewRow();
    row["Name"] = "Fisher Pete";
    row["Phone"] = "401-609-7623";
    customers.Rows.Add(row);
    row = customers.NewRow();
    row["Name"] = "Brown Kelly";
    row["Phone"] = "803-438-2771";
    customers.Rows.Add(row);
}
```

Step 7. Adding slot for a report

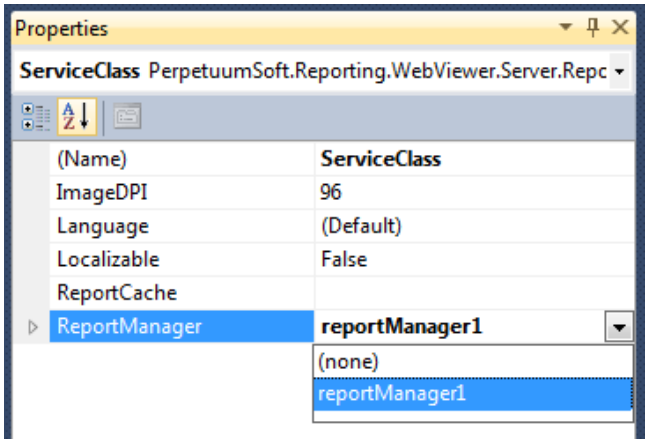
Now add the ReportManager component (double click on “ReportManager” in ToolBox); this component is responsible for report generation.



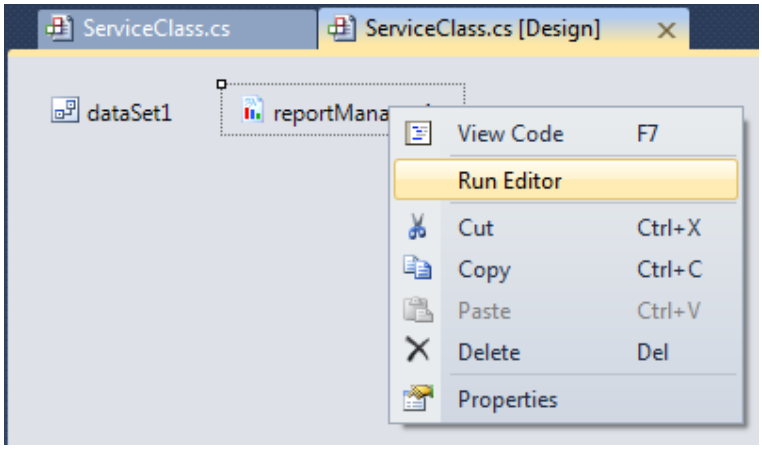
After that the ReportManager node (reportManager1) will appear in the designer.



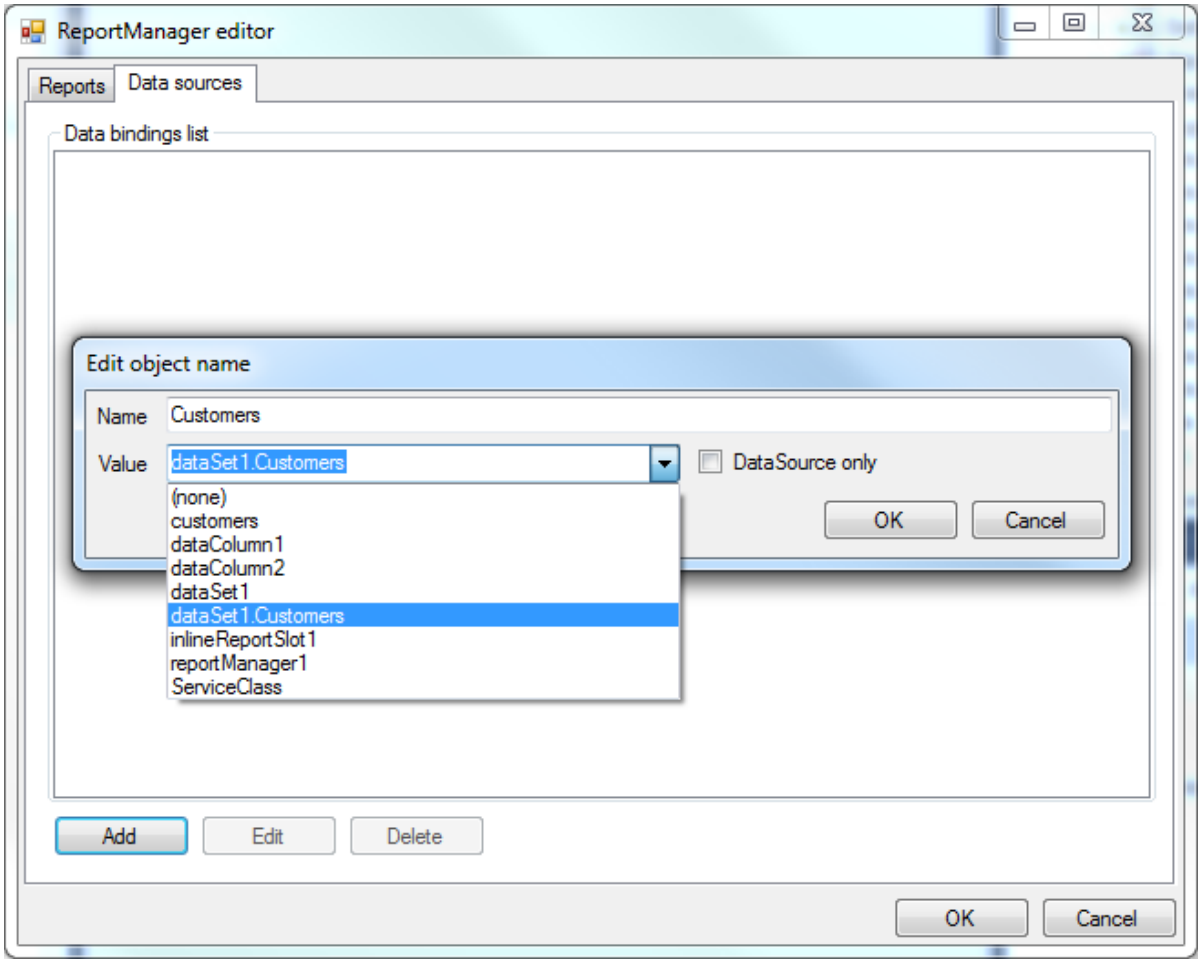
Set the ReportManager property of the ServiceClass class. In order to do that, open ServiceClass properties in the Properties window. Select reportManager1 from the list:



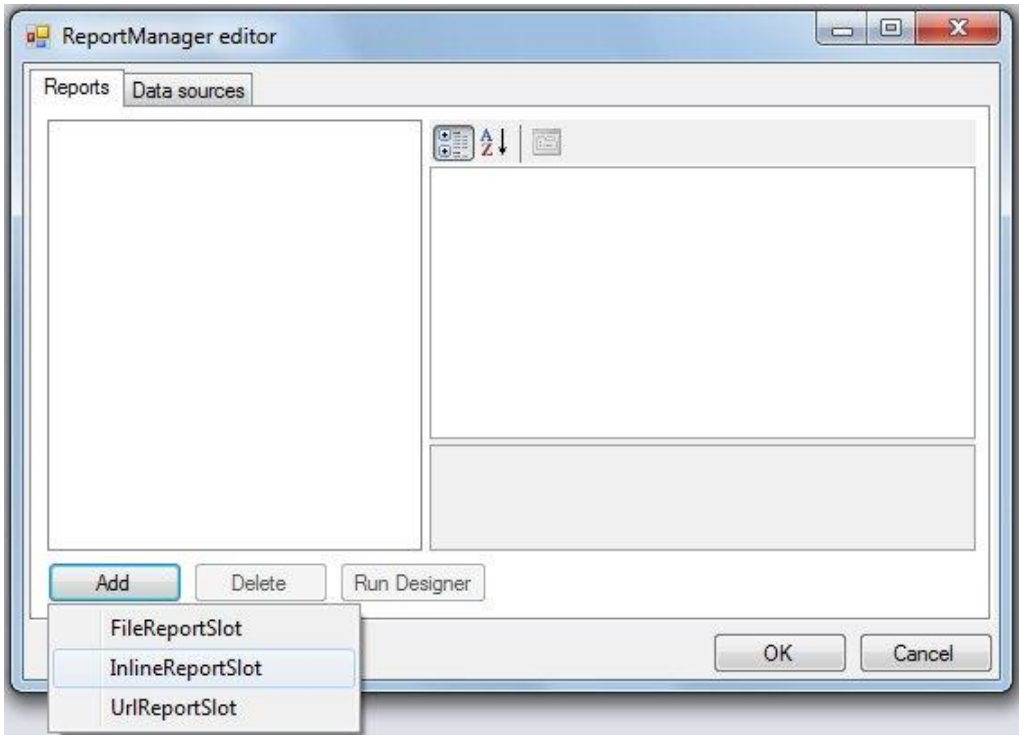
Run Report Manager Editor by right-click on reportManager1 and choosing Run Editor



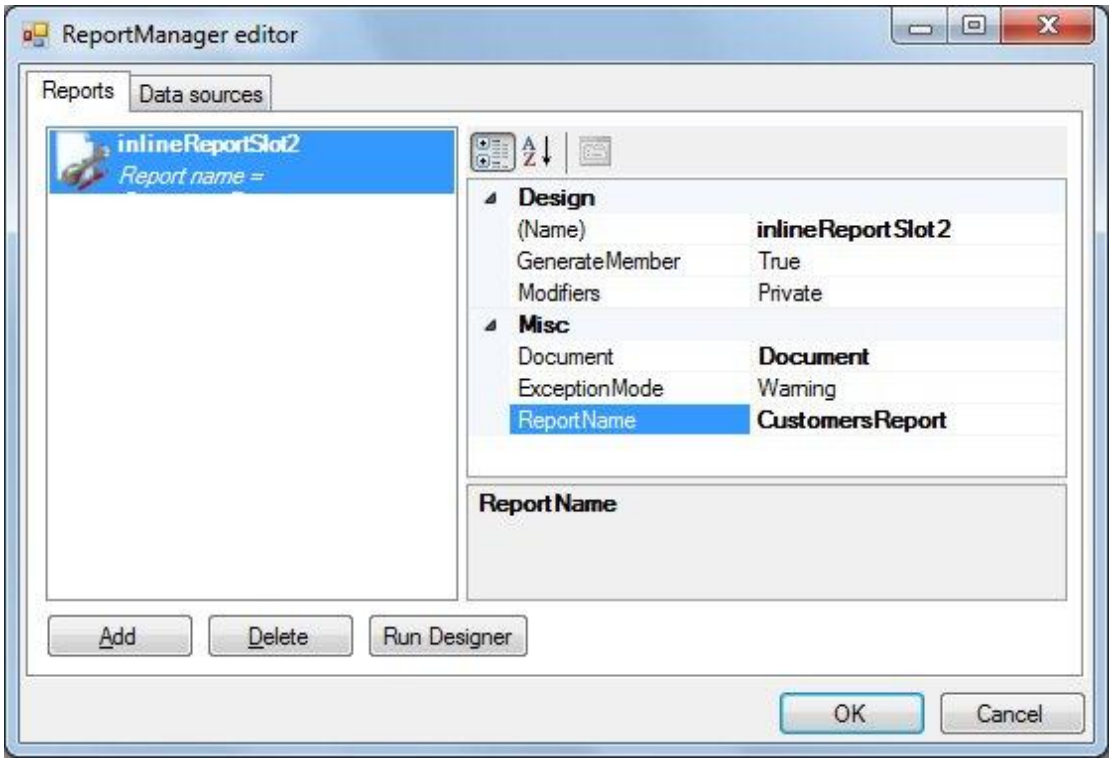
Before you start creating report template add a data source by which the report will be generated. Add Customers table to the Data binding list located on the “Data sources” tab (click “Add” button, in appeared form “Edit object name” set Name value to Customers, select dataSet1.Customers in the combo box list).



In the “Reports” tab, add a new object – “InlineReportSlot” by clicking the “Add” button.

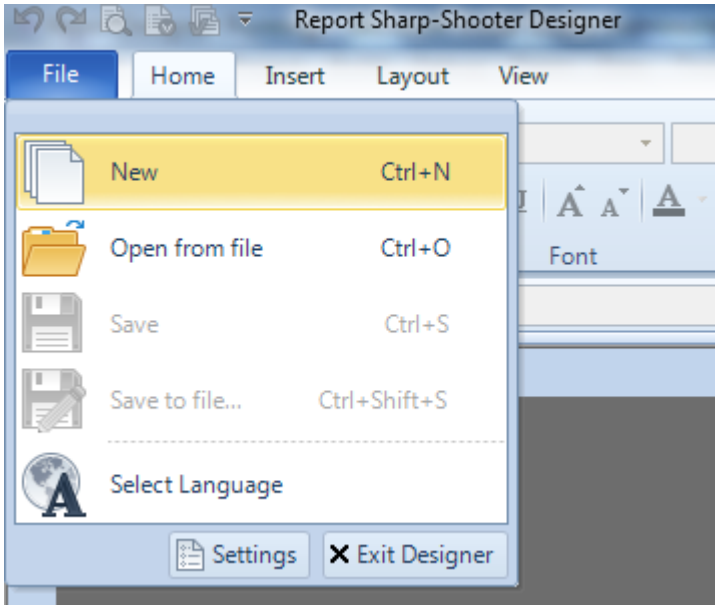


Set the ReportName property value to CustomersReport. Afterwards you will get the required document from the Report Manager exactly by that name. Then press the “Run Designer” button to launch report designer.

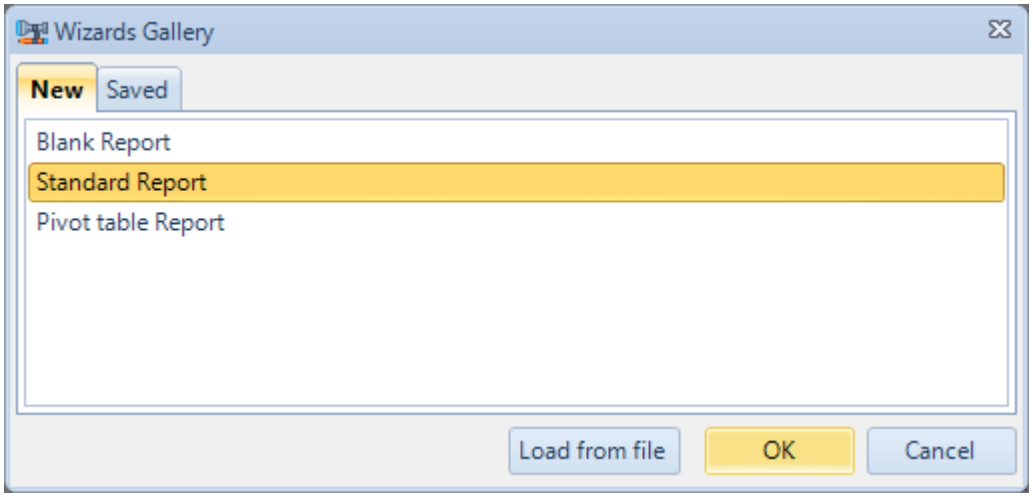


Step 8. Creation of a report with Wizard

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

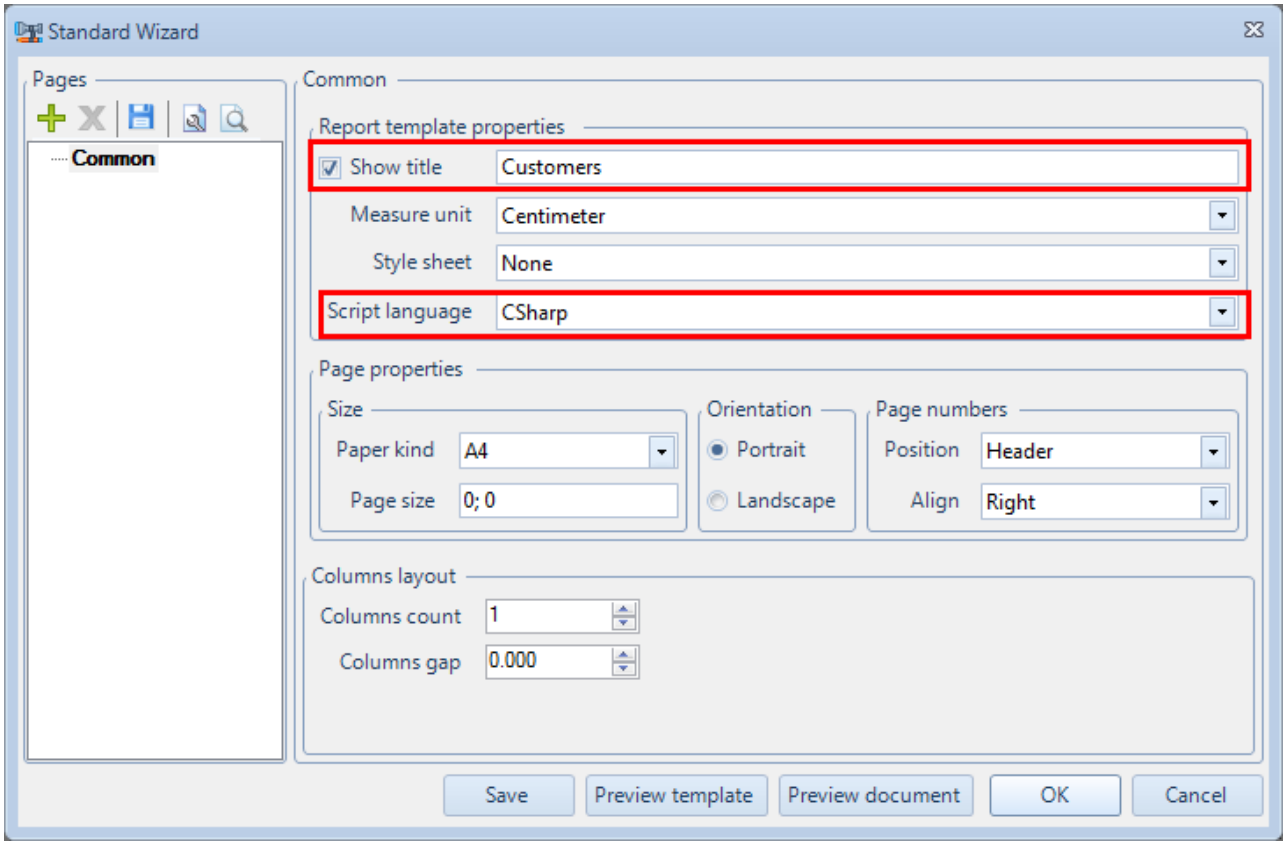


Select Standard Report in the list under the “New” tab and press OK.

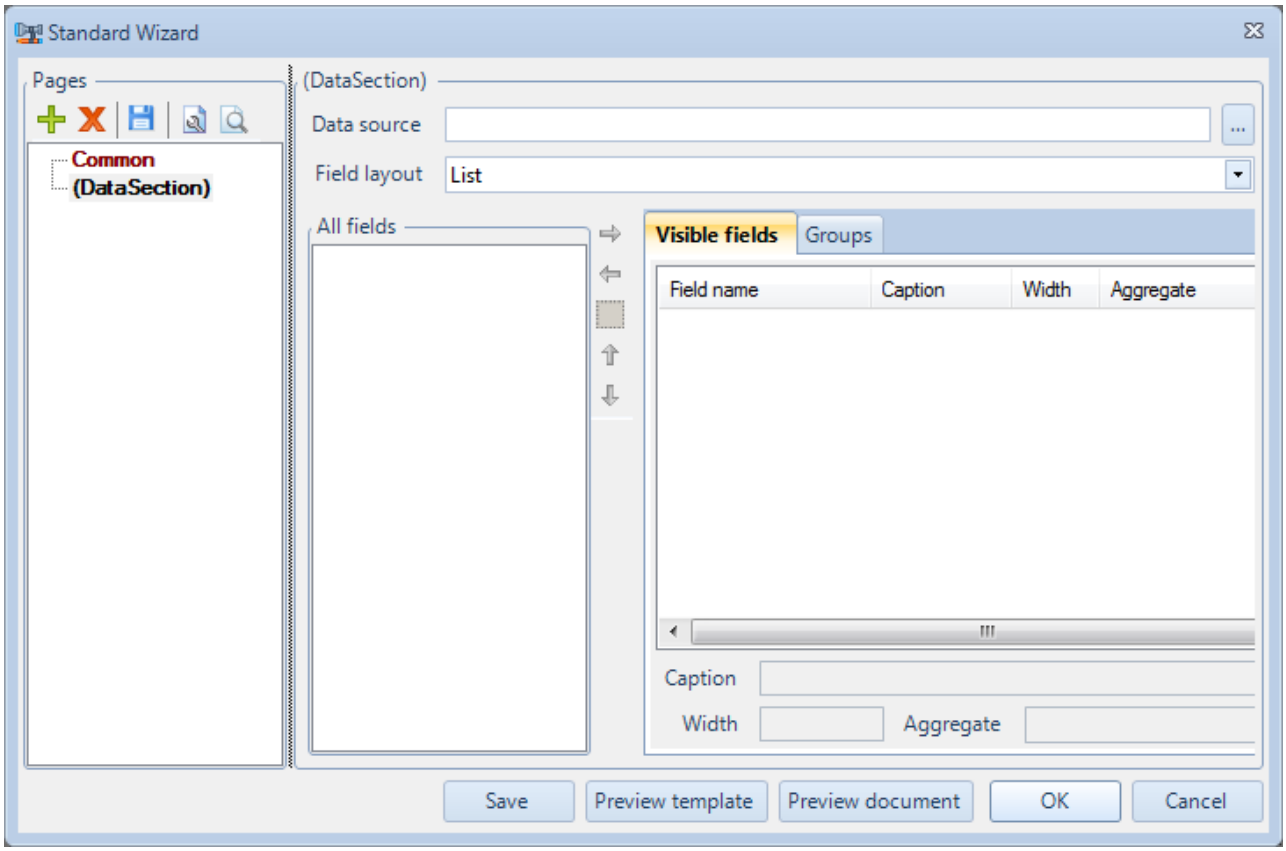


The Standard Wizard window will appear on the screen.

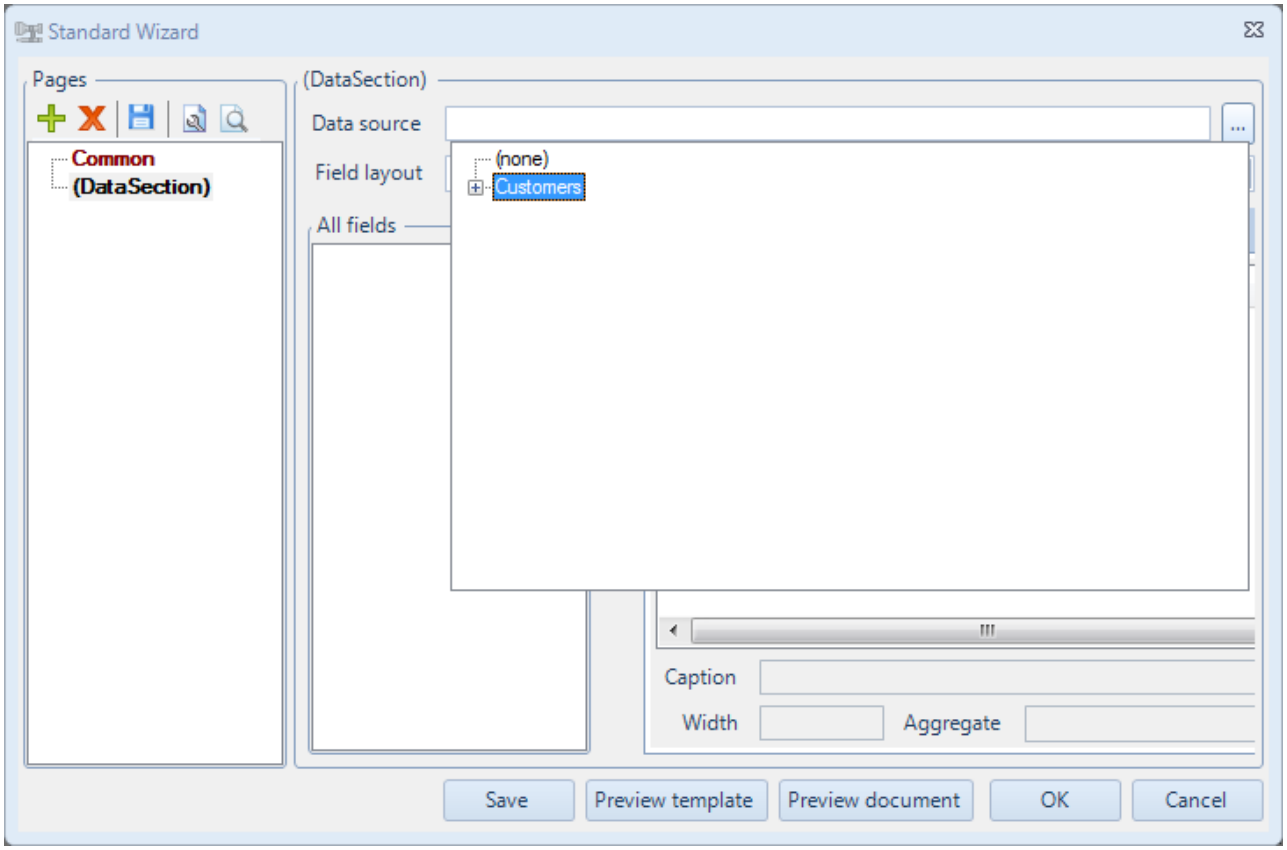
Set document parameters as shown in the following figure.



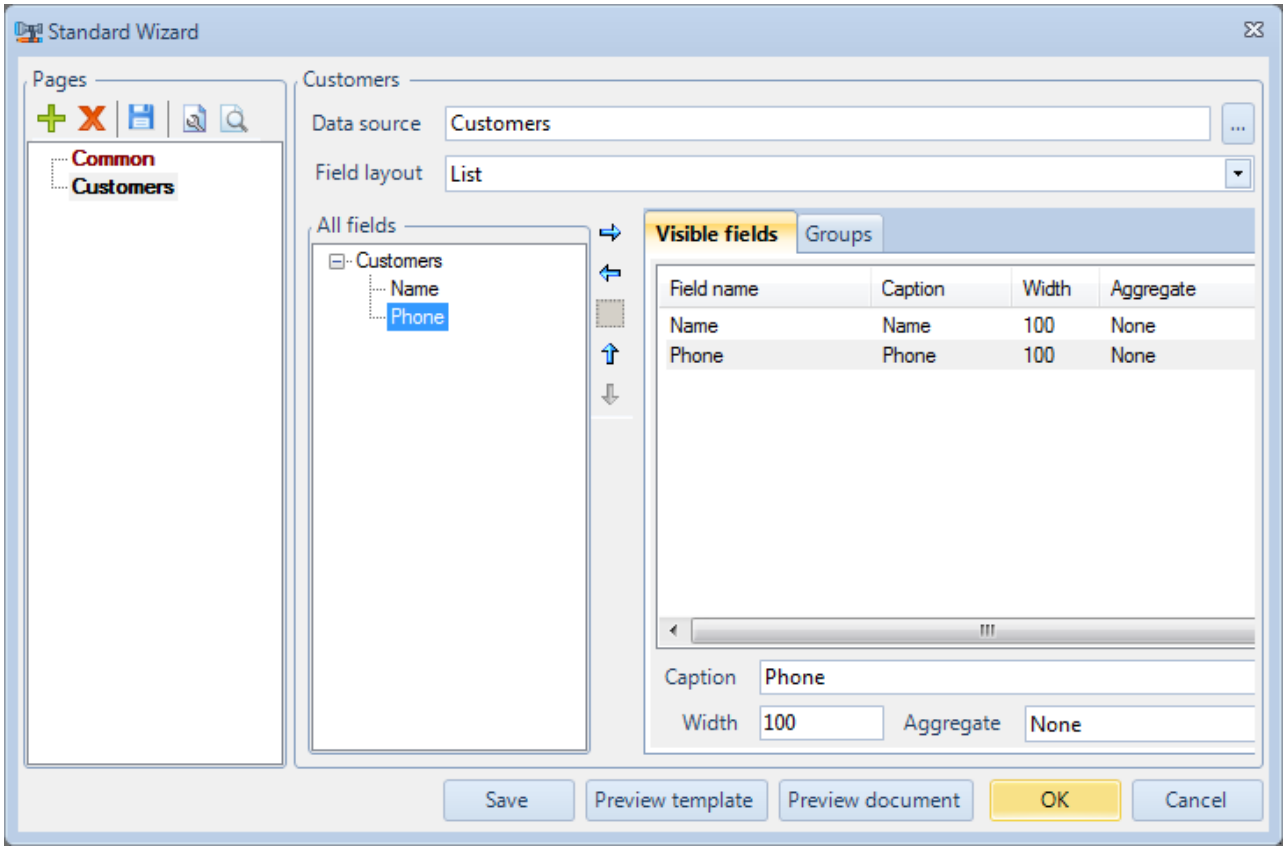
Add data source using the “Add” button ( ).



Click  button, in appeared tree view select “Customers” by double click.



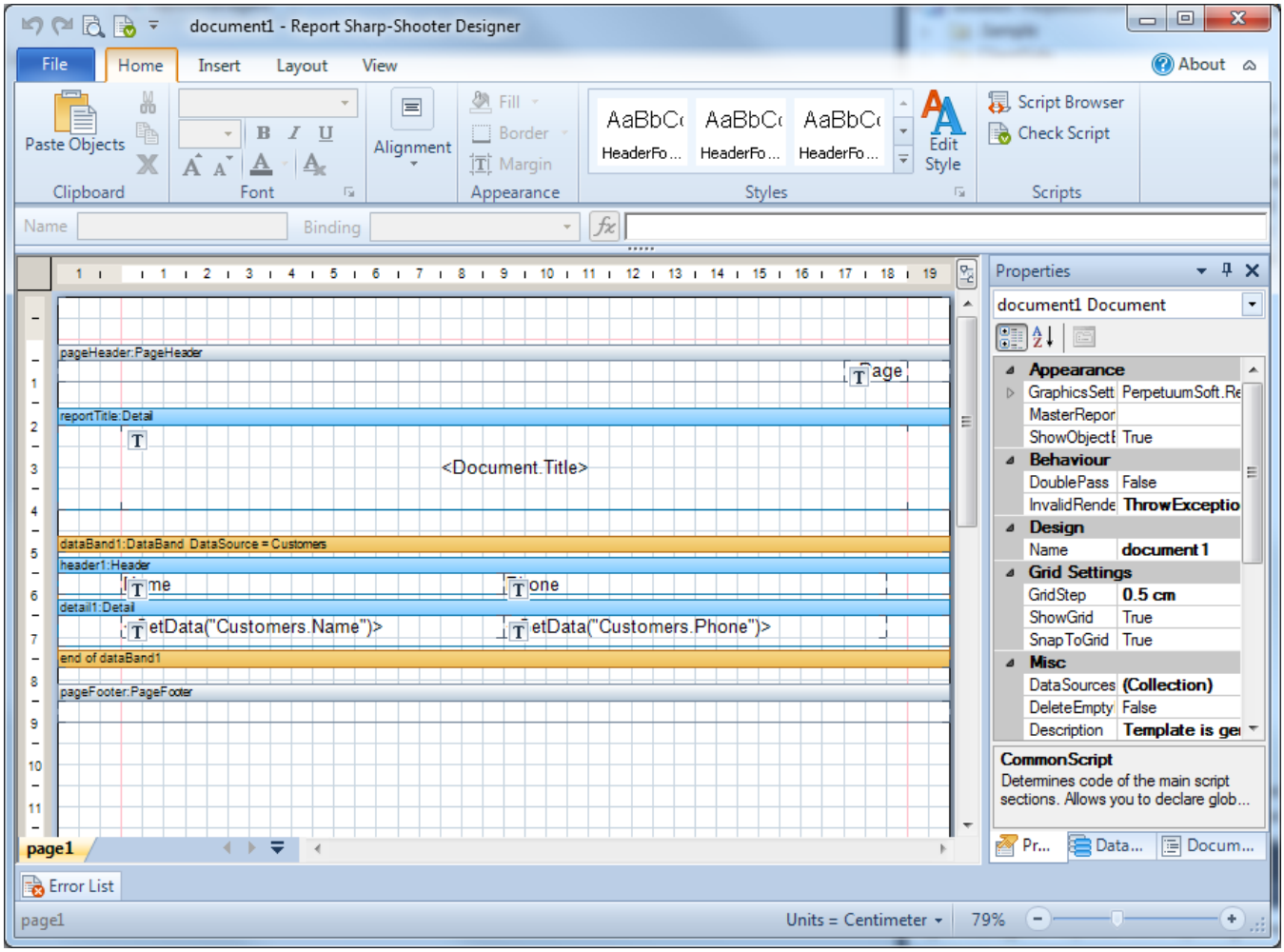
Select fields you want to output in the report (you should move both fields Name and Phone):



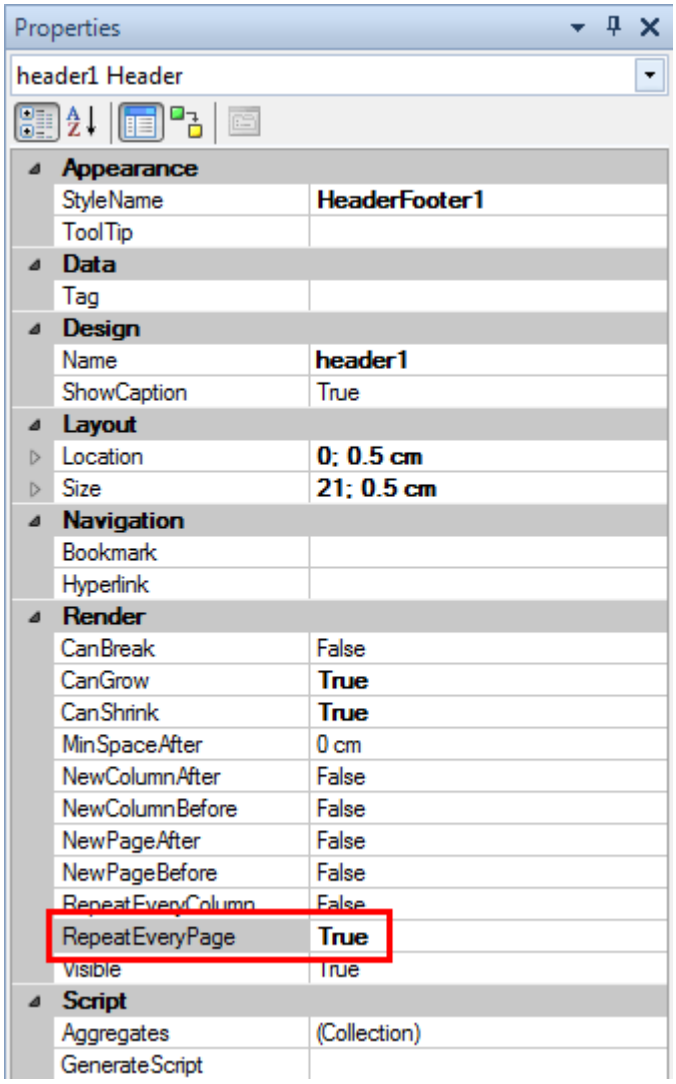
Press “OK”.

Step 9. Setting up a report

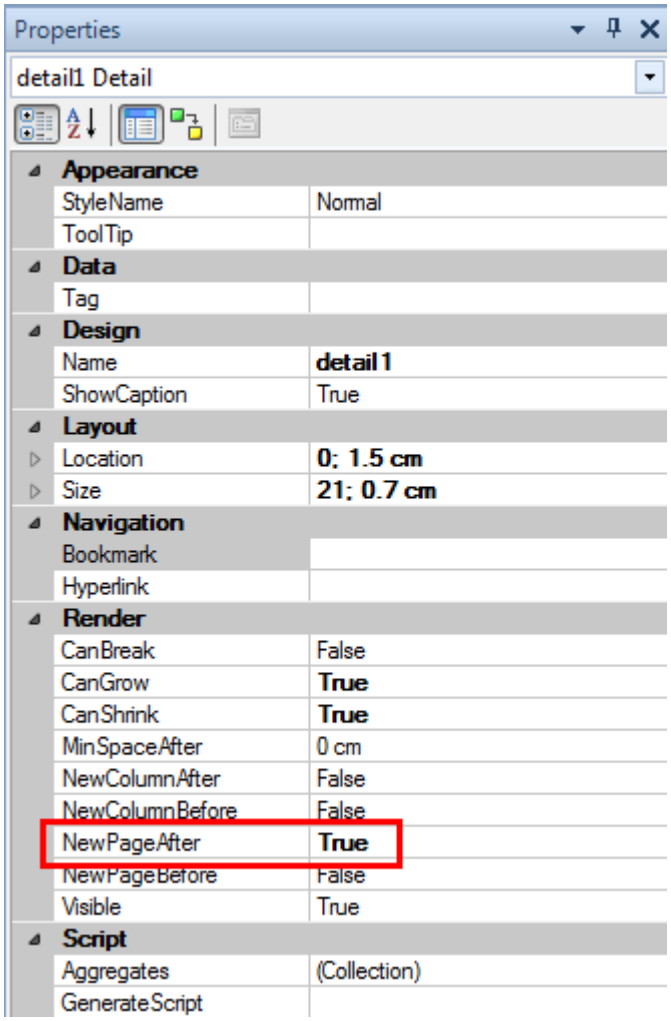
The template is created.



Set the “RepeatEveryPage” property to True for the header1 element in the “Properties” window.



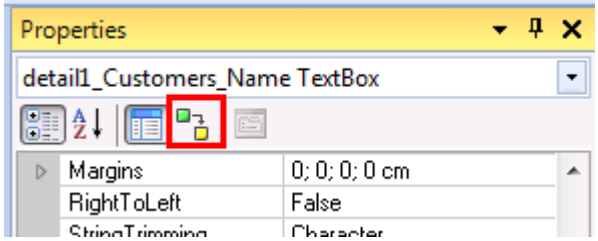
Set the “NewPageAfter” property to True for the detail1 element in the “Properties” window.



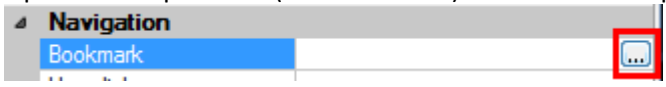
Step 10. Adding navigation

Let's add navigation to a report. Bookmarks tree will be used for navigation through the report.

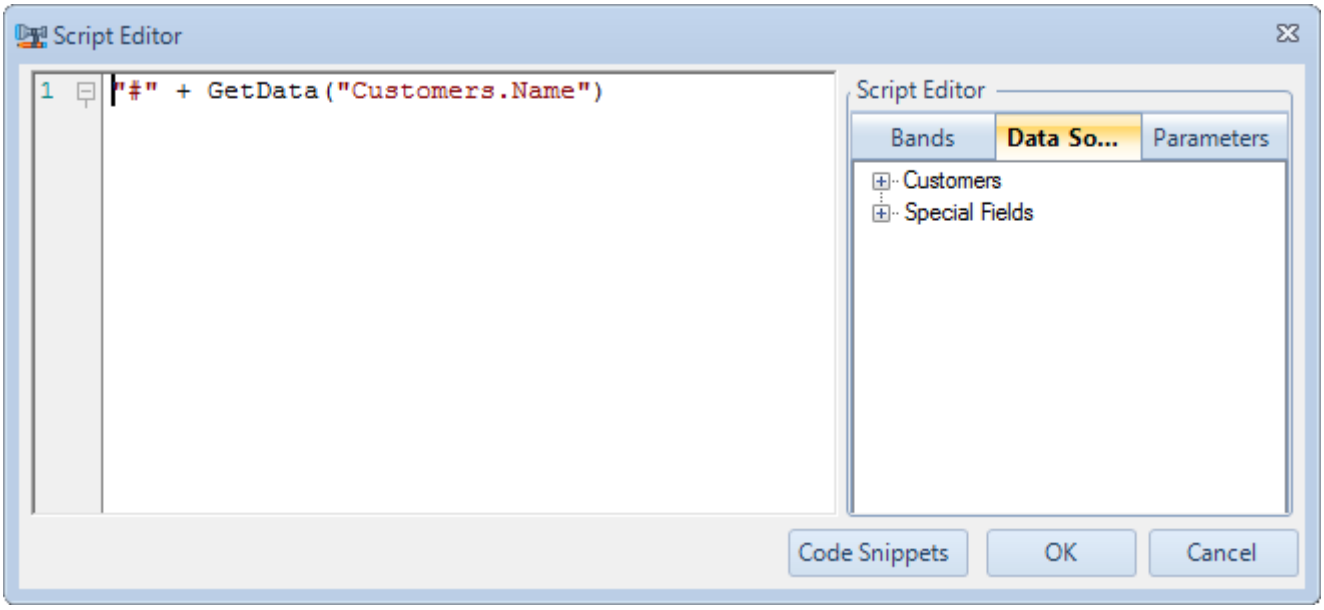
Select "Bindings" for the detail1_Customers_phone element in the "Properties" window.



Open the script editor (the ... button) for "Bookmark" property.



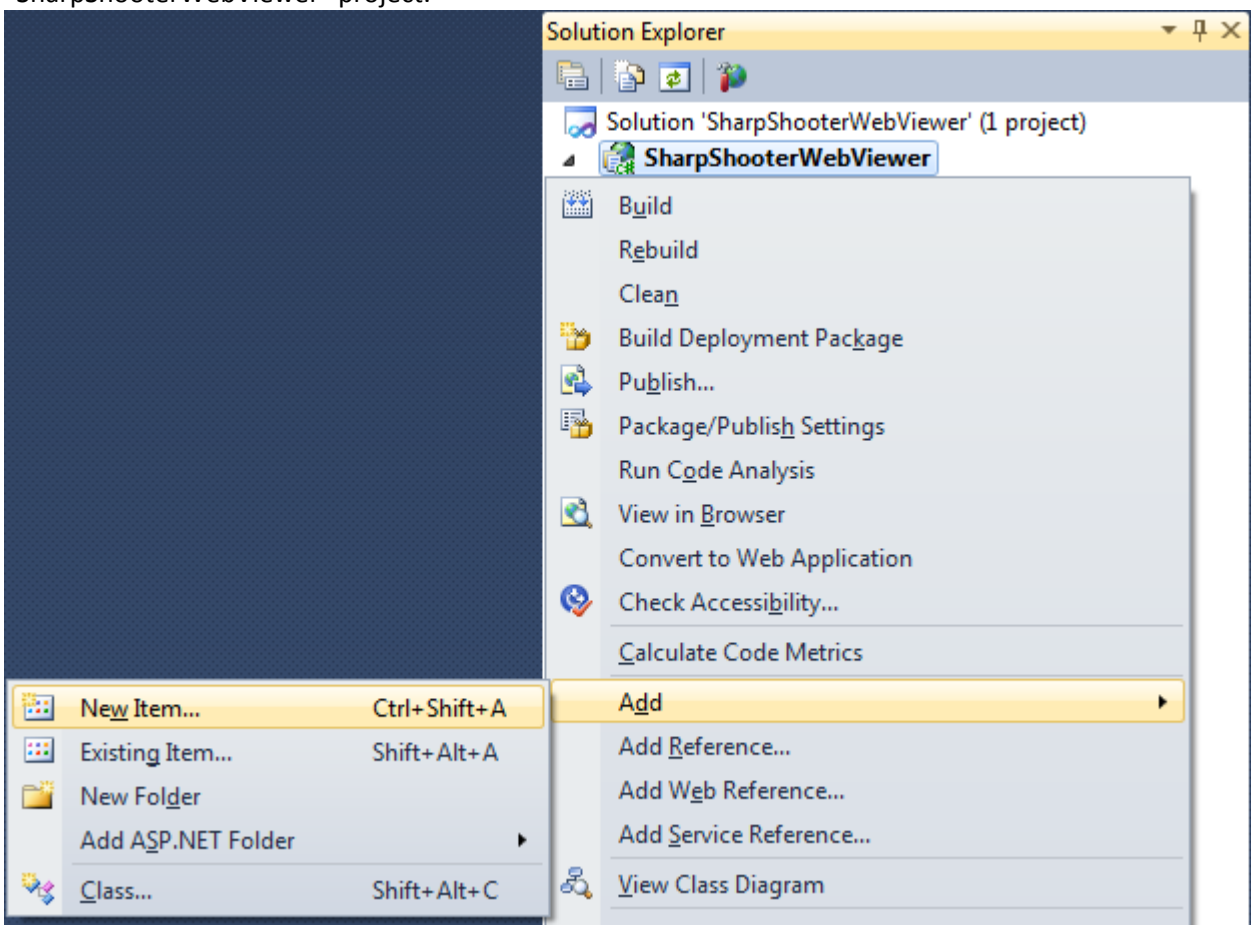
Enter "#" + in the script editor. Expand the "Customer" data source in the "Data Source" tab. Double click on the "Name" element in order to add the script code which will get customer name.



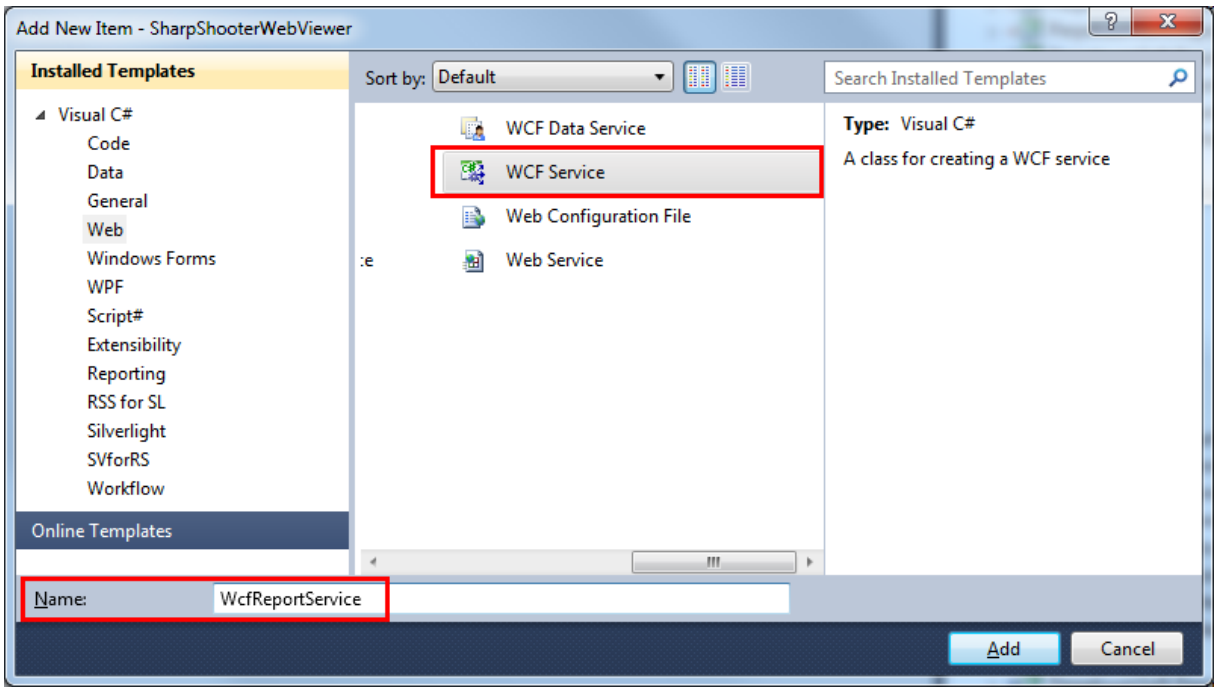
Now you should save template and close designer.

Step 11. Adding WCF service

Let's add WCF service to the project. Select the "Add\New Item..." item in the context menu of the "SharpShooterWebViewer" project.



Select the "WCF Service" element. Set the "WcfReportService" name in the "Name" field.

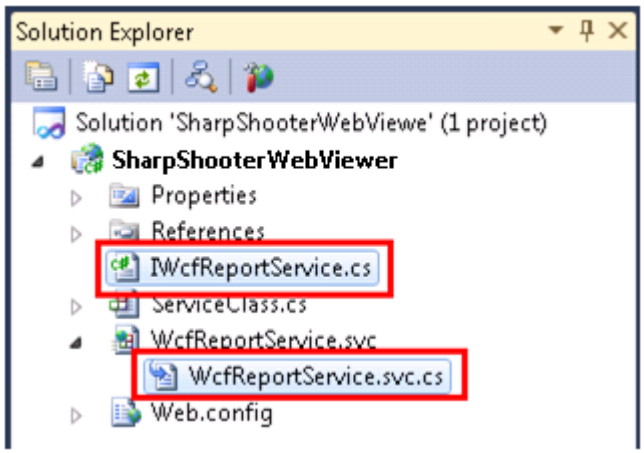


Remove the following files from the project:

IWcfReportService.cs – the interface which defines the service contract;

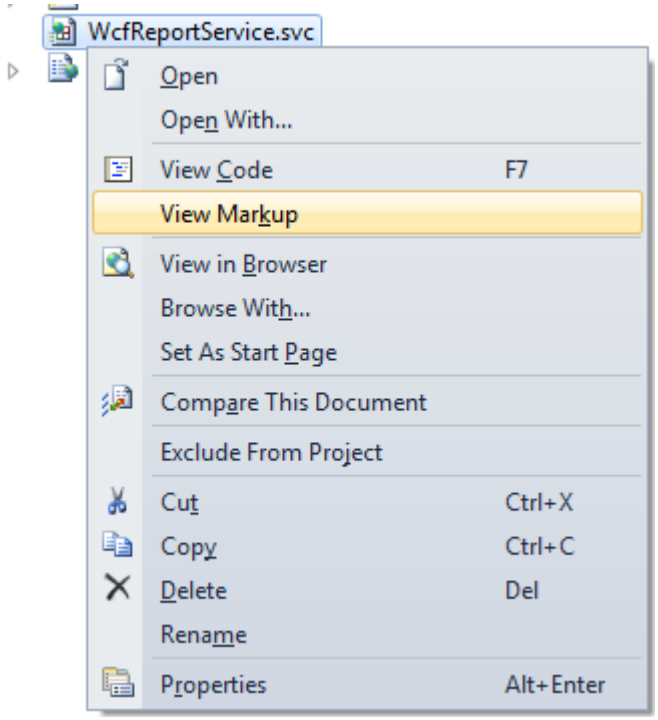
WcfReportService.svc.cs – the class which implements the logic of service work.

You don't need them since the created "ServiceClass" implements the logic of service work.



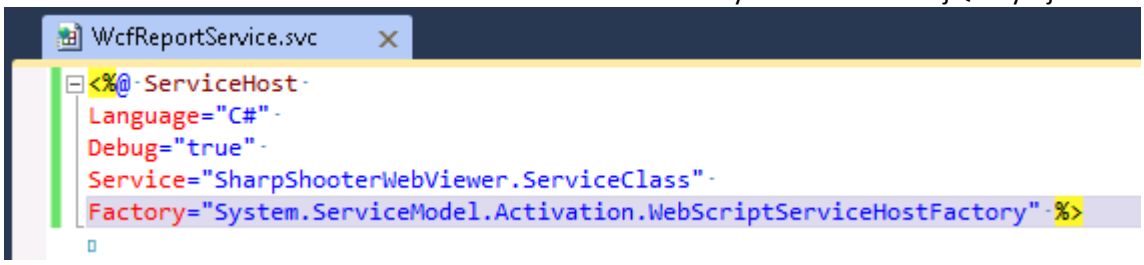
Step 12. Setting up service for the work with data in the Json format

Open the markup file of the service. Select the "WcfReportService.svc" file in the "Solution Explorer" window. Select "View Markup" in the context menu.



Since the logic of the service implementation is in the separate file, you can remove information about CodeBehind.

Set the full name of the class, which implements the logic of the service work, in the “Service” attribute. Add the “Factory” attribute with the “System.ServiceModel.Activation.WebScriptServiceHostFactory” value to add the Json serialization. This serialization is needed for the work of your service with jQuery Ajax.



Open the web.config file (double click on the web.config file in the “Solution Explorer”). Find the “system.serviceModel” section in the web.config file. This section is added by default when you add WCF service to the project. The section looks as follows:

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="">
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <serviceHostingEnvironment multipleSiteBindingsEnabled="true" />
</system.serviceModel>
```

Add the “aspNetCompatibilityEnabled” attribute with the value equal to true to the “serviceHostingEnvironment” section. Thus, Wcf service will work in the ASP.NET context. Add section which describes the behavior of the added service to the “serviceBehaviors” section. You can do it as follows:

```
        <behavior name="WcfReportServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
```

Add the “service” section to the “system.serviceModel” section. Add the “service” section, which contains settings for your service, to the “services” section. It can be done as follows:

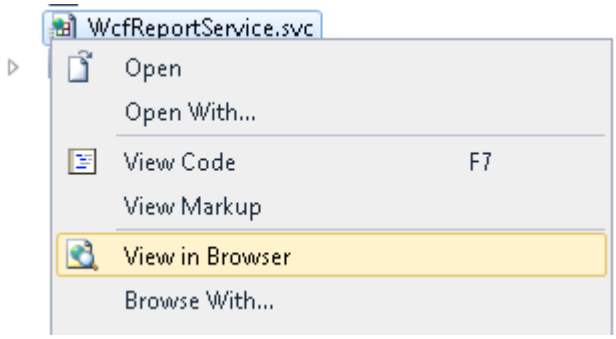
```
    <services>
      <service name="SharpShooterWebViewer.WcfReportService" behaviorConfiguration="WcfRepo
rtServiceBehavior">
        <endpoint address="" binding="wsHttpBinding" contract="PerpetuumSoft.Reporting.WebV
iewer.Server.IReportService" />
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
      </service>
    </services>
```

The “system.serviceModel” section should look as follows:

```
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
        <behavior name="WcfReportServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true" /
>
    <services>
      <service name="SharpShooterWebViewer.WcfReportService" behaviorConfiguration="WcfReportServiceB
ehavior">
        <endpoint address="" binding="wsHttpBinding" contract="PerpetuumSoft.Reporting.WebViewer.Serv
er.IReportService" />
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
      </service>
    </services>
  </system.serviceModel>
```

Step 13. Checking if the set service is available

Select the “WcfReportService.svc” file in the “Solution Explorer”, select the “View in Browser” item in the context menu.



You should see the following page in the browser:

If you don't see this page in the browser then there is some error in the service setting. Please revise steps 1-13.

If you see this page then the setting of the server part of the application is finished. Let's set up the client application.

Step 14. Adding script files

Add the following files to the project:

jquery-1.5.1.js – jQuery plugin;

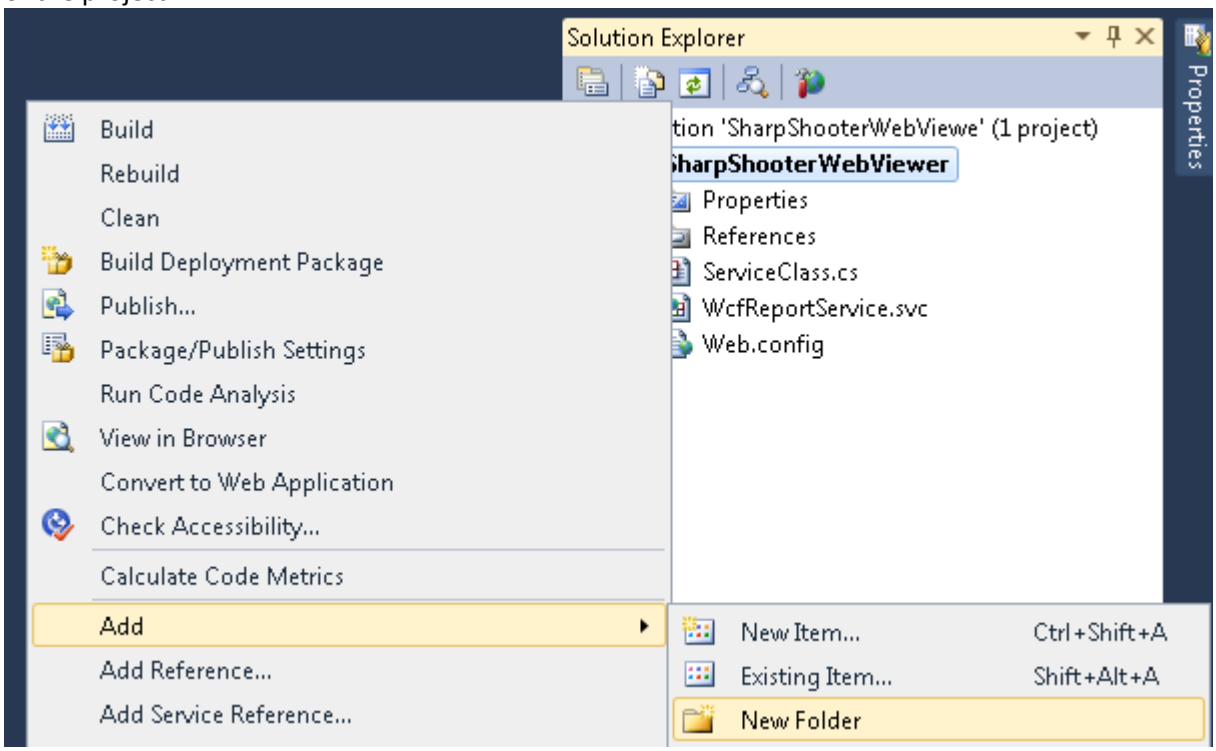
jquery.treeview.js – implements the work of bookmarks tree;

mscorlib.js – provides functionality like a type system in script, and basic utility APIs (you need it when you use the classes created with Script#);

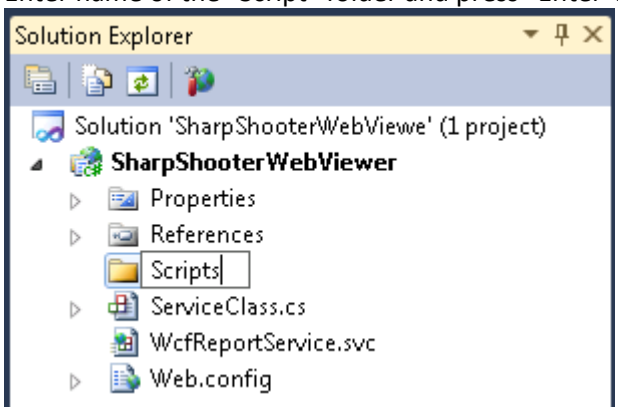
PerpetuumSoft.Reporting.WebViewer.Client.js – contains the classes which implement logic of getting and displaying reports;

PerpetuumSoft.Reporting.WebViewer.Client.Model.js – the classes of data model.

Add the folder, which will contain scripts, to the project. Select the “Add\New Folder” item in the context menu of the project.



Enter name of the “Script” folder and press “Enter”.



Select the “Scripts” folder in the“Solution Explorer”. Select “Add”->”Existing Item…” in the context menu.

Add the following files from the “Web\Scripts” folder:

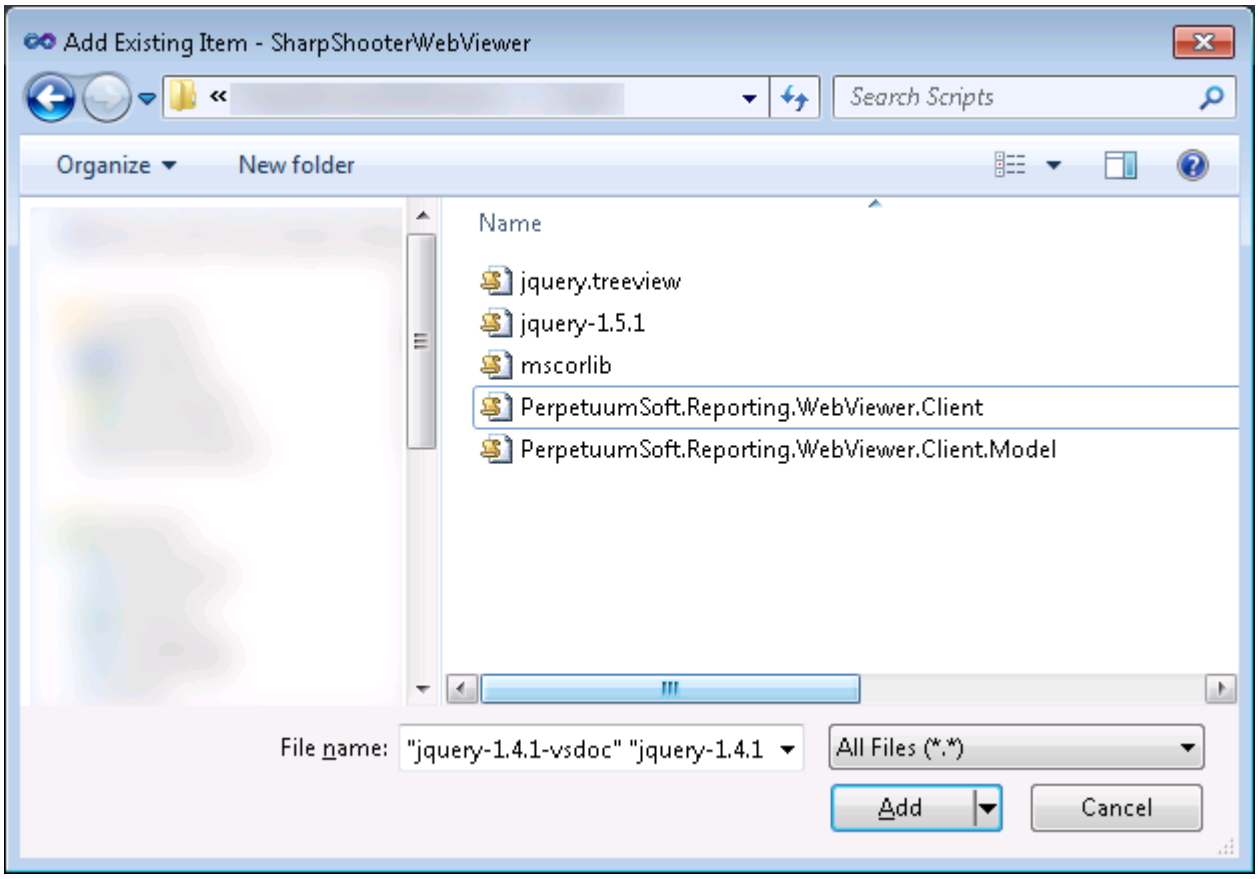
jquery-1.5.1.js

jquery.treeview.js

mscorlib.js

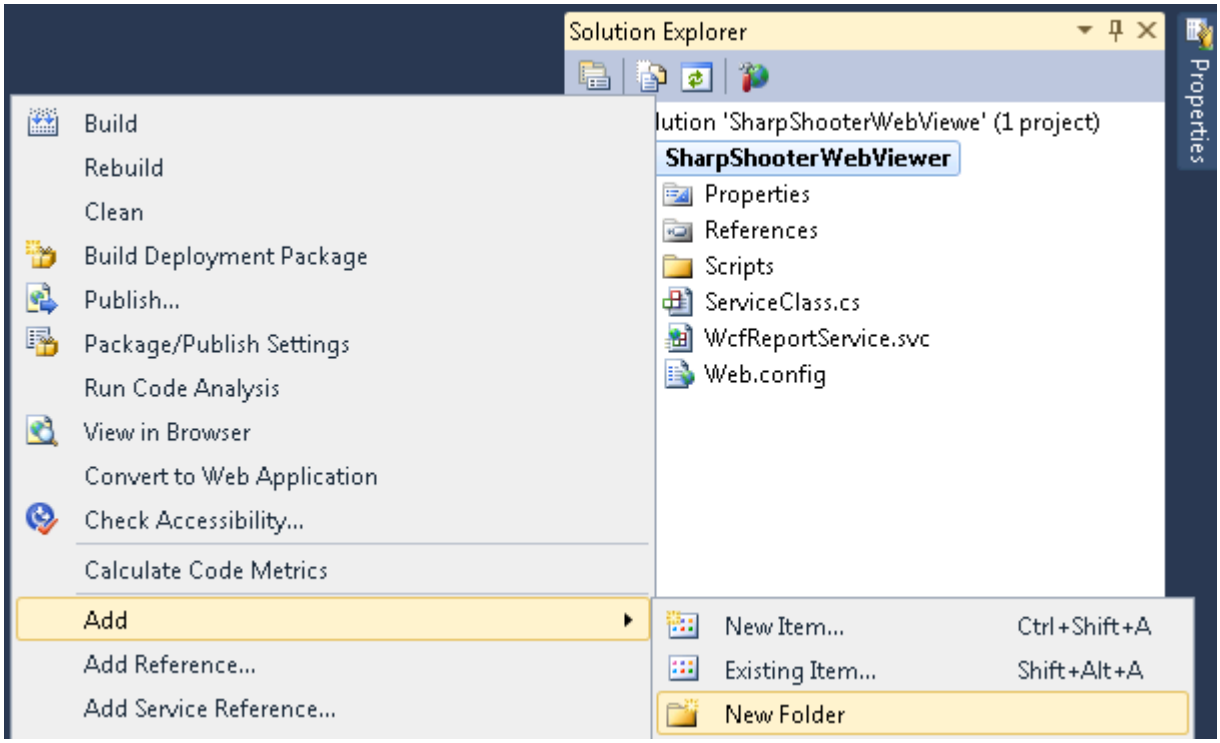
PerpetuumSoft.Reporting.WebViewer.Client.Model.js

PerpetuumSoft.Reporting.WebViewer.Client.js

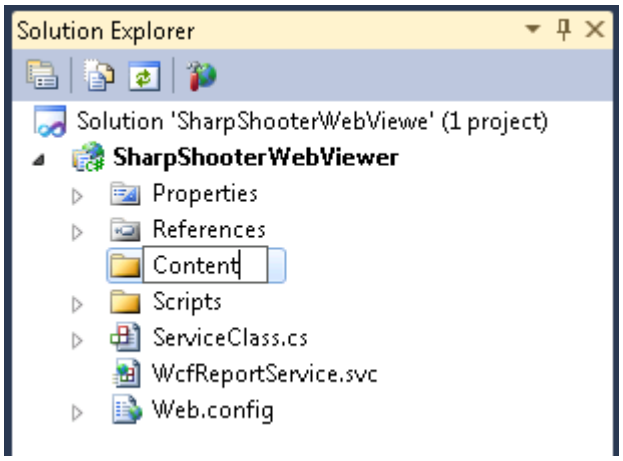


Step 15. Adding styles

Add the “Content” folder to the project. This folder will contain files with styles. Select the “Add\New Folder” item in the context menu.



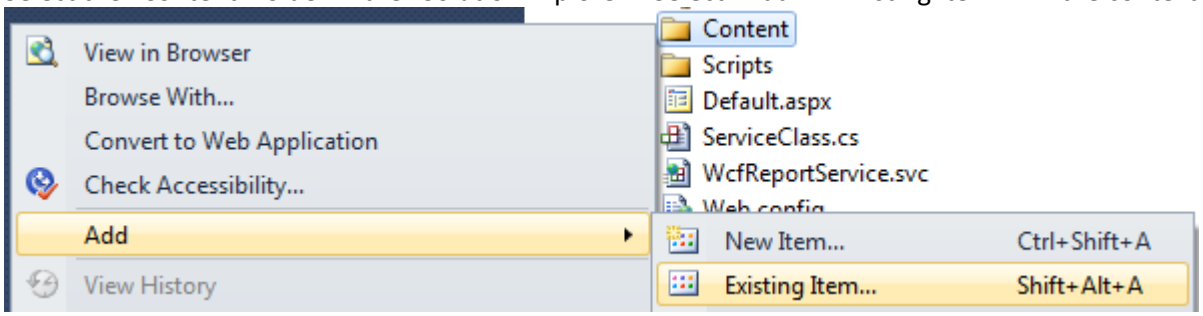
Enter name of the “Content” folder:



Add the style needed for correct report display to the project. It is necessary to add the following file with styles to the project:

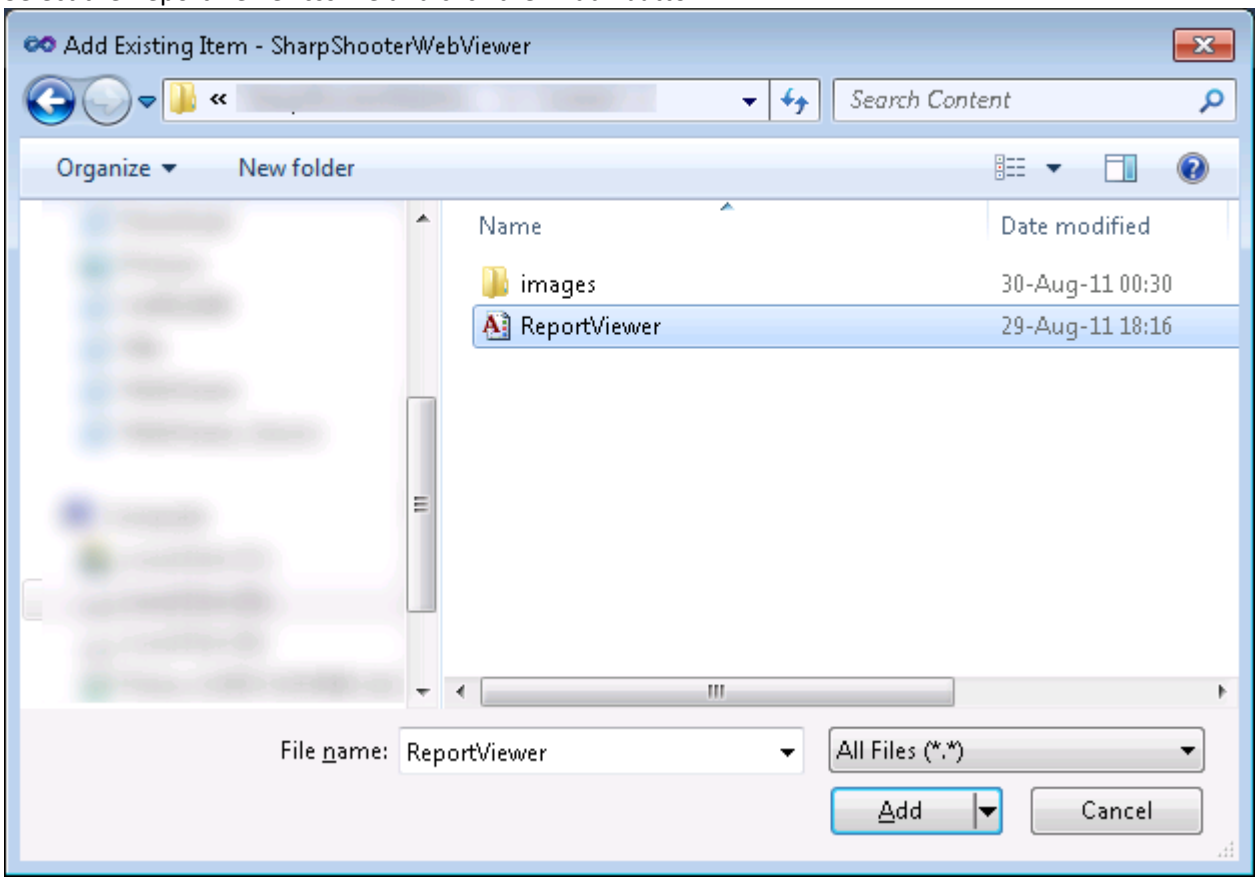
ReportViewer.css – it describes styles which are used for report display.

Select the “Content” folder in the “Solution Explorer”. Select “Add”->”Existing Item…” in the context menu.



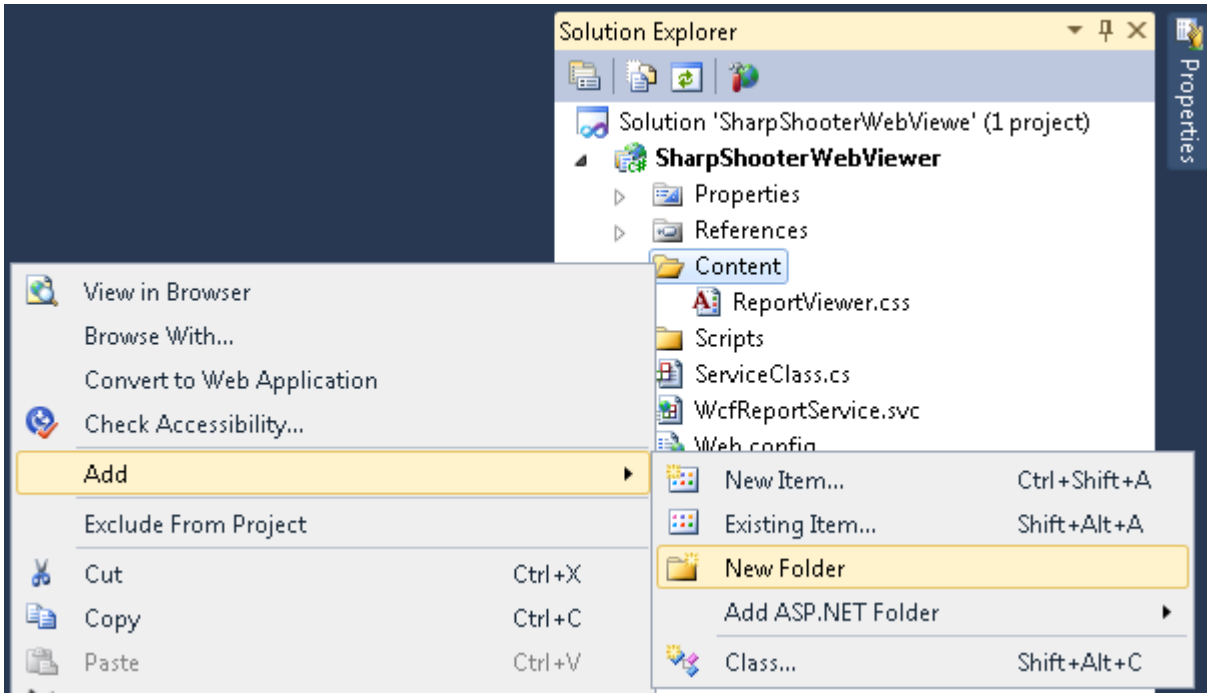
Add the ReportViewer.css file from the “Web\Content” folder.

Select the ReportViewer.css file and click the “Add” button.

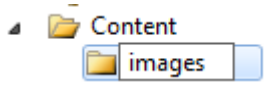


Step 16. Adding images

You should add images files. These images will be used for display of bookmarks tree. Add the folder for storing images to the project. Select the “Content” folder in the “Solution Explorer”. Select “Add\New Folder” in the context menu.



Set the “images” name for the folder in the “Solution Explorer”:



Add images to the “images” folder. Select the added “images” folder. Select “Add\Existing Item…” in the context menu.

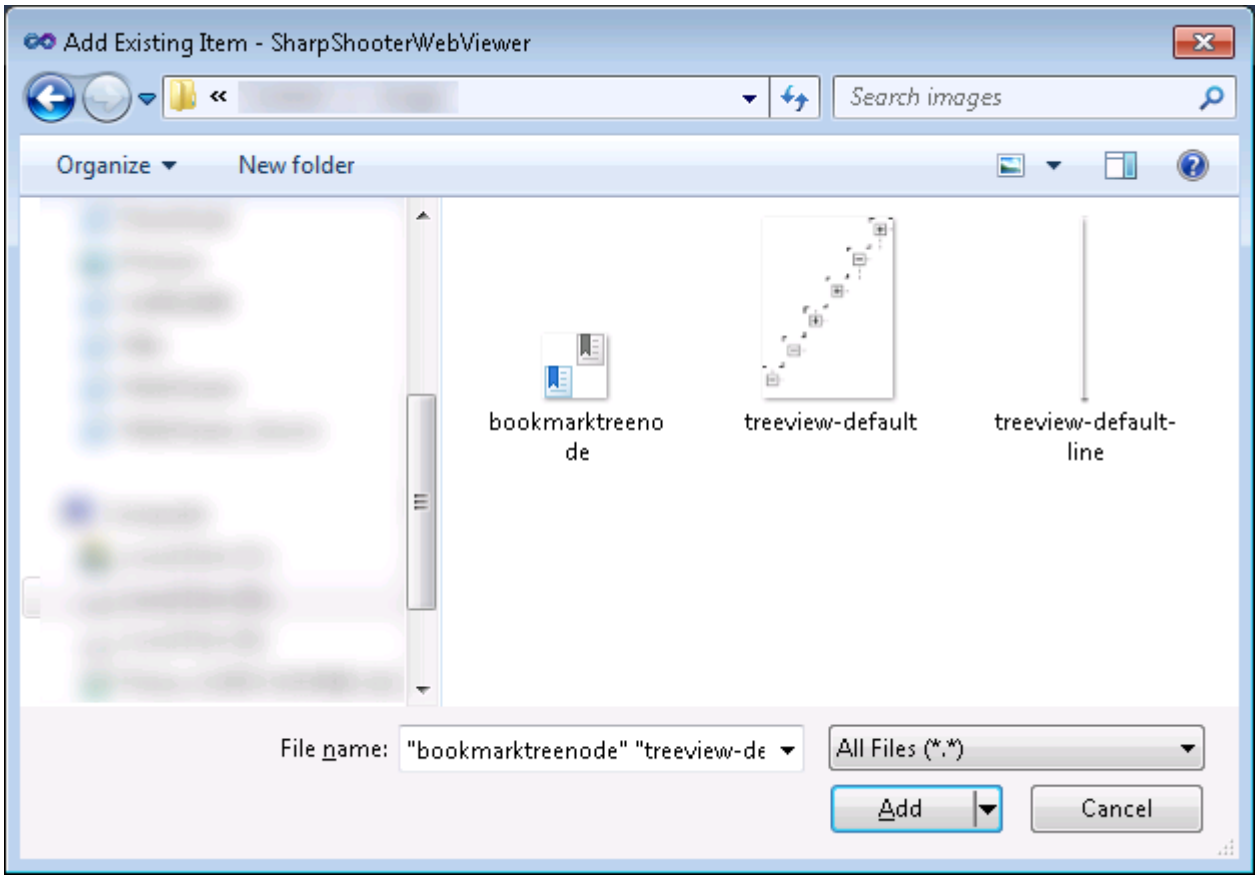
Navigate to the “Web\Content\images” and select the following images files:

bookmarktreenode.png;

treeview-default.gif;

treeview-default-line.gif.

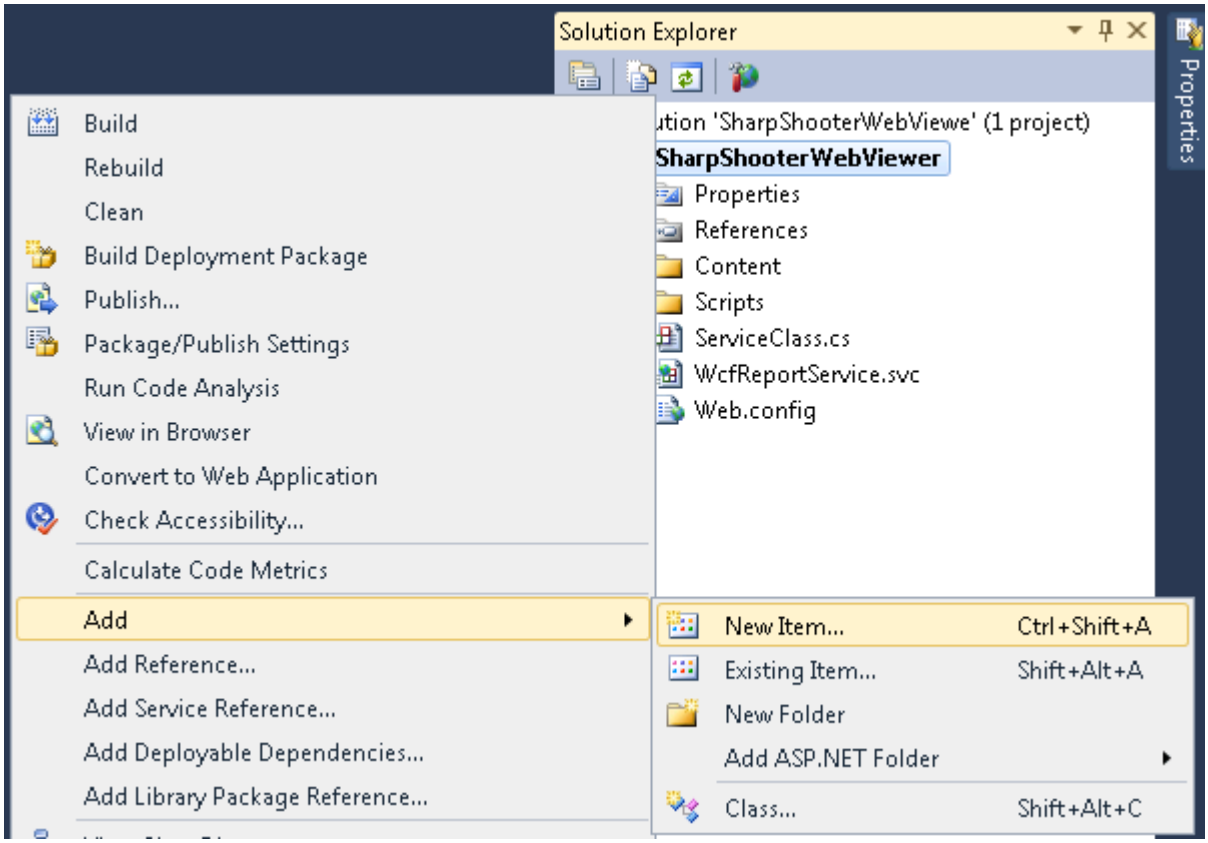
Click the “Add” button.



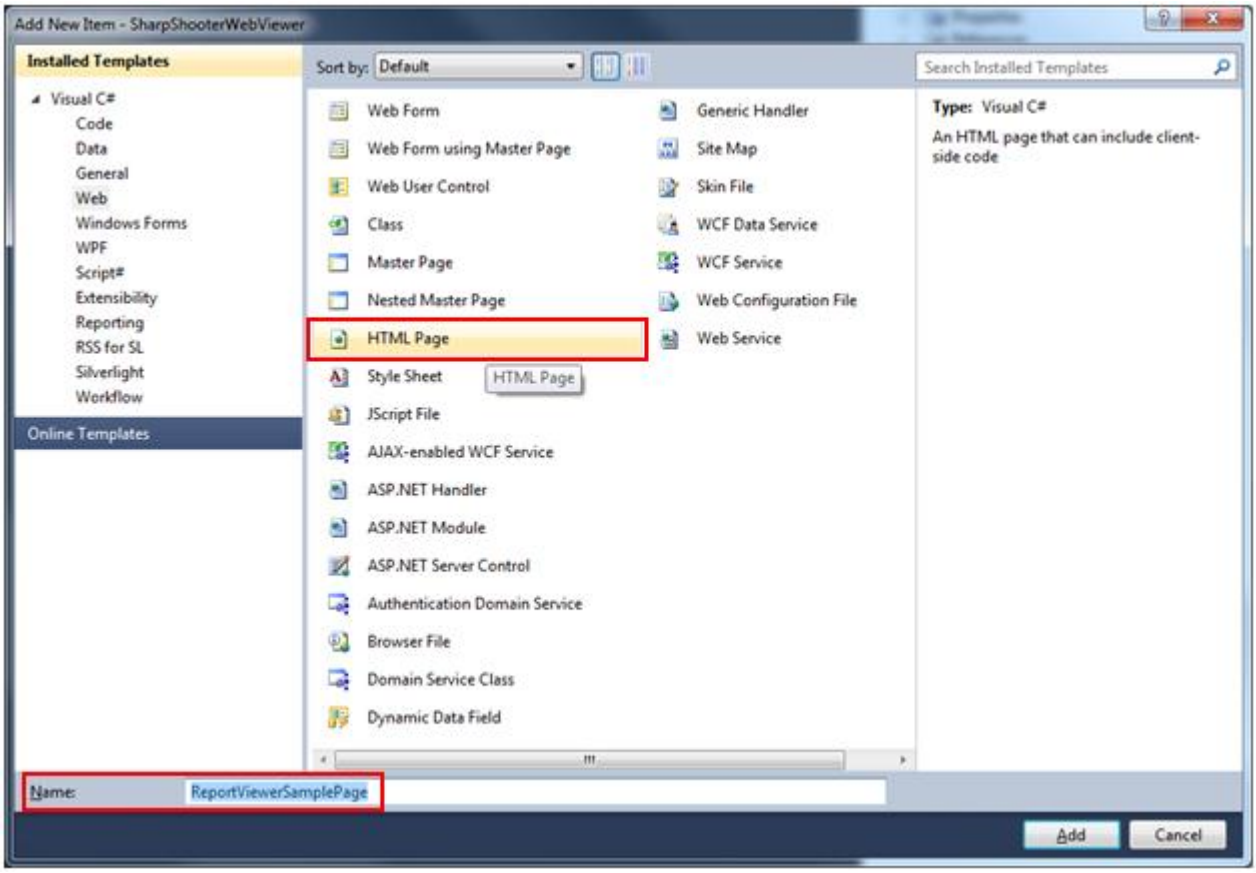
Let’s create a page which will contain report viewer after you add all the needed files to the project.

Step 17. Adding Html page to the project

Select the SharpShooterWebViewer project in the “Solution Explorer”. Select “Add\New Item…” in the context menu.



Select the “HTML Page” element in the opened window. Set “ReportViewerSamplePage” in the “Name” field.



Step 18. Adding scripts and styles to the page

You need to add styles and scripts to the Html page (on ReportViewerSamplePage.htm) to be able to use them. Transfer the necessary files to the Html page (on ReportViewerSamplePage.htm) from the “Solution Explorer”.

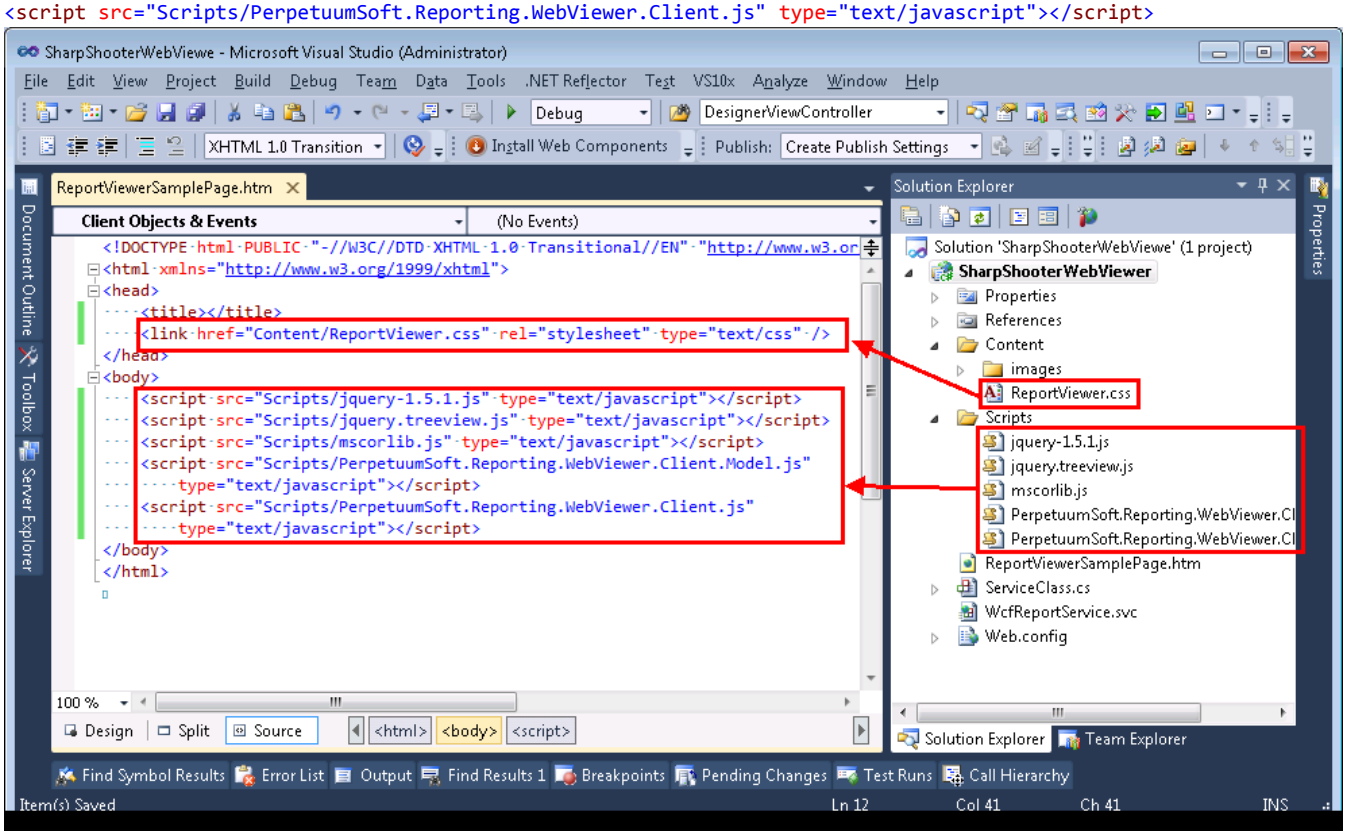
Remove “http://localhost:5555/” in the paths of the added files.

The code for adding files with styles should look as follows:

```
<link href="Content/ReportViewer.css" rel="stylesheet" type="text/css" />
```

The code for adding files with scripts should look as follows:

```
<script src="Scripts/jquery-1.5.1.js" type="text/javascript"></script>
<script src="Scripts/jquery.treeview.js" type="text/javascript"></script>
<script src="Scripts/mscorlib.js" type="text/javascript"></script>
<script src="Scripts/PerpetuumSoft.Reporting.WebViewer.Client.Model.js" type="text/javascript"></script>
<script src="Scripts/PerpetuumSoft.Reporting.WebViewer.Client.js" type="text/javascript"></script>
```



Step 19. Displaying reports on a web page

Add the div element to the “body” section on the web page. This element displays a report:

```
<div id="ReportViewerElement">
</div>
```

Set the identifier in the div element. This identifier is needed for getting the div element from the javascript code.

You need to add the javascript code to the page. This code loads a document from server:

```
<script type="text/javascript">
   $(document).ready(function () {
```

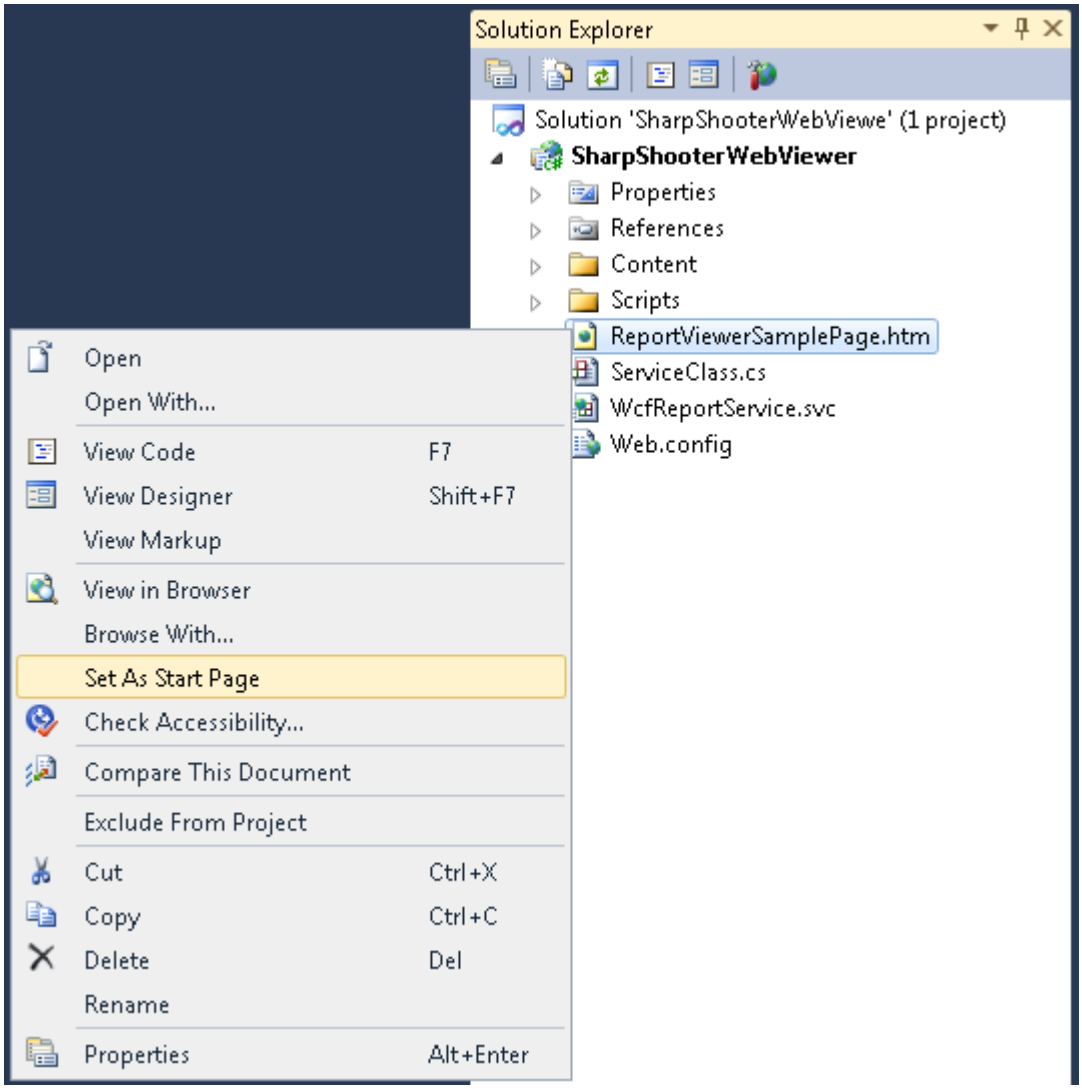
```
            var reportViewer = new PerpetuumSoft.Reporting.WebViewer.Client.ReportViewer("#ReportViewerElement");
            reportViewer.setServiceUrl("http://localhost:5555/WcfReportService.svc");
            reportViewer.reportName = "CustomersReport";
            reportViewer.renderDocument();
            reportViewer.setThumbnailsControl("#thumbanailsView");
            reportViewer.setDocumentMapControl("#documentMapView");
        });
    </script>
```

The class object is created in the code. This object loads a document and displays it on the page. When you create the object, you should indicate the web page element which will be used for report display. Then, the service address and report name are set. The renderDocument method initializes the loading of a document from the server.

The code of the ReportViewerSamplePage.htm should look as follows:

```
<head>
    <title></title>
    <link href="Styles/ReportViewer.css" rel="stylesheet" type="text/css" />
    <link href="Styles/jquery.treeview.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <script src="Scripts/jquery-1.5.1.js" type="text/javascript"></script>
    <script src="Scripts/jquery.treeview.js" type="text/javascript"></script>
    <script src="Scripts/json2.js" type="text/javascript"></script>
    <script src="Scripts/mscorlib.js" type="text/javascript"></script>
    <script src="Scripts/PerpetuumSoft.Reporting.WebViewer.Client.Model.js"
        type="text/javascript"></script>
    <script src="Scripts/PerpetuumSoft.Reporting.WebViewer.Client.js"
        type="text/javascript"></script>
        <script type="text/javascript">
        $(document).ready(function () {
            var reportViewer = new PerpetuumSoft.Reporting.WebViewer.Client.ReportViewer("#ReportViewerElement");
            reportViewer.setServiceUrl("http://localhost:5555/WcfReportService.svc");
            reportViewer.reportName = "CustomersReport";
            reportViewer.renderDocument();
            reportViewer.setThumbnailsControl("#thumbanailsView");
            reportViewer.setDocumentMapControl("#documentMapView");
        });
    </script>
    <div id="ReportViewerElement">
    </div>
</body>
```

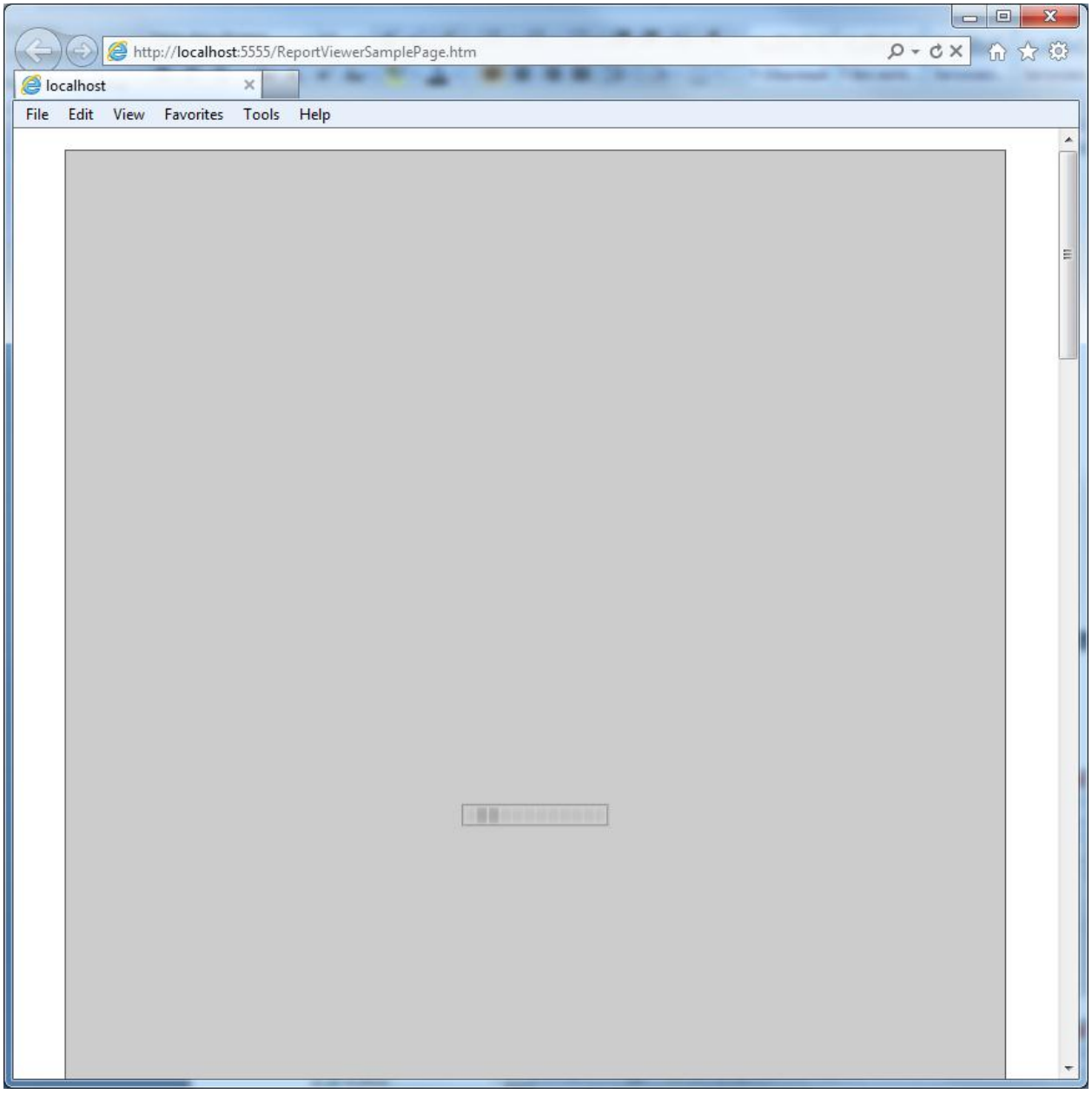
Select the “ReportViewerSamplePage.htm” file in “Solution Explorer”. Select “Set As Start Page” in the context menu. Thus, you set the “ReportViewerSamplePage.htm” page as a start page of your web application. This page will be opened when you run the application.



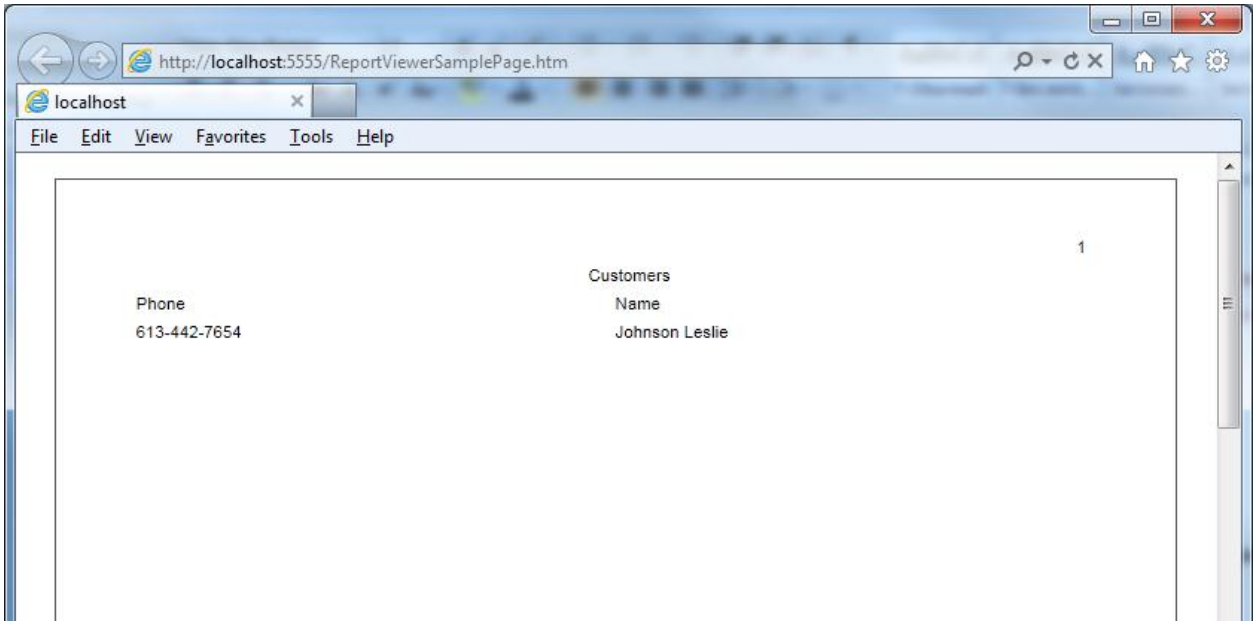
Launch application by clicking the “Start Debugging” button on the main Visual Studio toolbar.

You will see throbber when you run the application in your web browser:

The area for the document display appears when the data about the document is loaded. After this, the application defines what pages will be visible and sends request for loading of visible pages. The progress bar (as it is shown on the screenshot below) will be displayed during the pages loading:



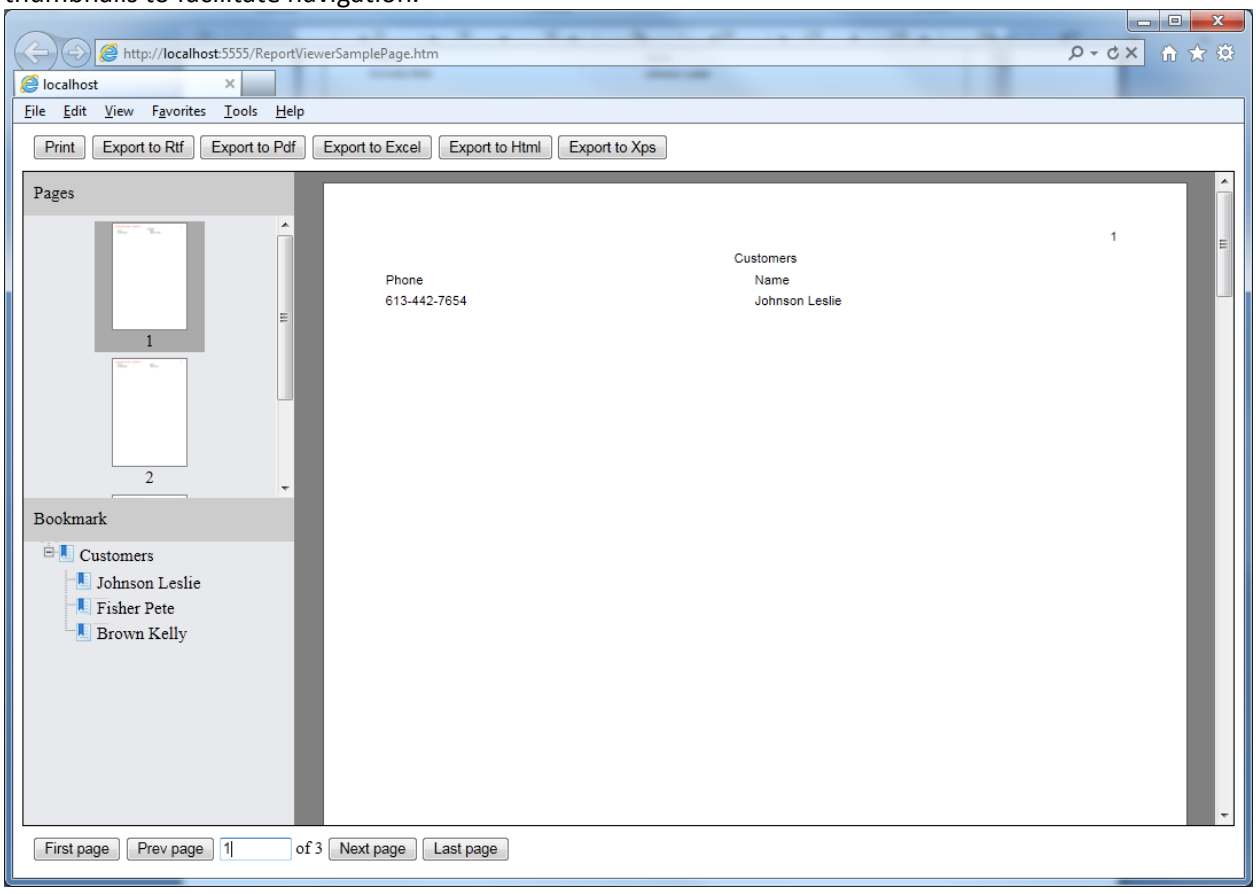
The page is displayed in the browser when it has been loaded:



Add the elements, which invoke methods for navigation through the report, report export and printing, to the page.

Step 20. Setting up appearance

Let's examine the following example. The example includes print button, buttons for document export, buttons for navigation to pages, number of pages and number of a current page. We added bookmarks tree and thumbnails to facilitate navigation.



Let's examine how to create such example.

Step 21. Page markup

The markup of the page, which screenshot you can see above, looks as follows:

```
 <div style="margin: 10px">
    <input id="printButton" type="button" value="Print" class="ExportButton" />
    <input id="exportToRtfButton" type="button" value="Export to Rtf" class="ExportButton" />
    <input id="exportToPdfButton" type="button" value="Export to Pdf" class="ExportButton" />
    <input id="exportToExcelButton" type="button" value="Export to Excel" class="ExportButton" />
    <input id="exportToHtmlButton" type="button" value="Export to Html" class="ExportButton" />
    <input id="exportToXpsButton" type="button" value="Export to Xps" class="ExportButton" />
</div>
<div style="height: 600px; margin: auto; border: solid 1px black;">
    <div style="height: 600px; background-color: White; float: left; width: 250px;">
        <div style="height: 300px;">
            <div style="height: 30px; background-color: #CCC; padding: 10px 0px 0px 10px;">
                <span>Pages</span>
            </div>
            <div id="ssr_thumbnailContentPanel" style="width:250px; height: 260px;">
            </div>
        </div>
        <div style="height: 300px;">
            <div style="height: 30px; background-color: #CCC; padding: 10px 0px 0px 10px;">
                <span>Bookmark</span>
            </div>
            <div id="documentMapView">
            </div>
        </div>
    </div>
    <div id="ReportViewerElement" style="height: 600px; background-color: Gray; overflow: auto;">
    </div>
</div>
<div style="margin: 10px;">
    <input id="firstPage" type="button" value="First page"/>
    <input id="prevPage" type="button" value="Prev page" />
    <input id="currentPage" type="text" title="Current page" style="width: 60px;" />
    <span>of </span><span id="pageCount">0</span>
    <input id="nextPage" type="button" value="Next page" />
    <input id="lastPage" type="button" value="Last page" />
</div>
```

Create handlers for the following events of the “reportViewer” object:

documentInfoLoadedEvent – the event which occurs when data about the report is loaded. This event allows you to get a list of pages with their sizes;

currentPageChangedEvent – the event which occurs when the current page was changed. This event allows you to get the number of the current page;

errorEvent – the event which occurs when error occurs. This event allows you to get data about the error.

Set what elements of the web page you need to use for output of thumbnails and bookmarks tree:

```
reportViewer.setThumbnailsControl("#ssr_thumbnailContentPanel");
reportViewer.setDocumentMapControl("#documentMapView");
```

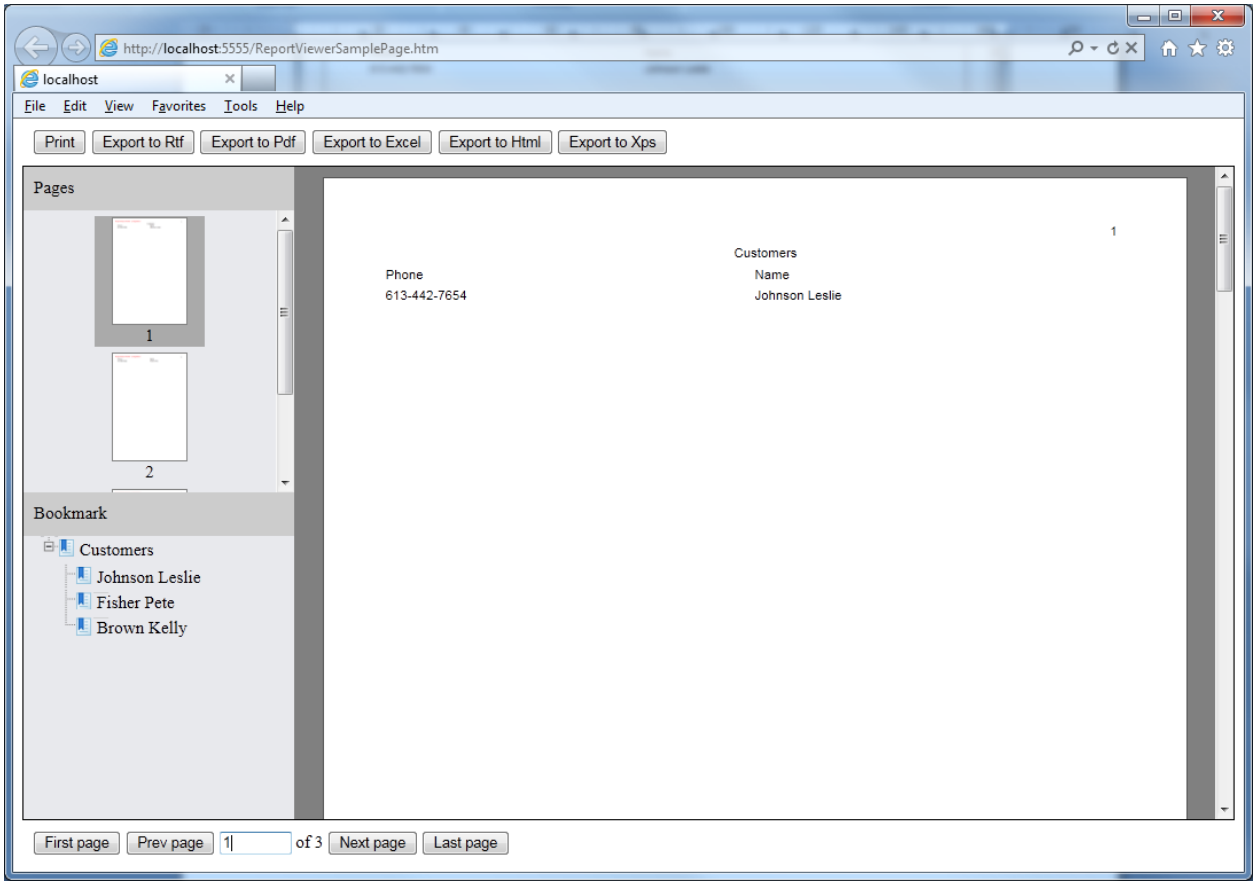
Add the handlers for print, export and navigation to pages buttons.

The javascript code looks as follows:

```
<script type="text/javascript">
```

```
    var reportViewer = null;
    $(document).ready(function () {
        Initialize();
        $("#printButton").click(function () {
            reportViewer.print();
        });
        $("#exportToRtfButton").click(function () {
            reportViewer.exportToRtf();
        });
        $("#exportToPdfButton").click(function () {
            reportViewer.exportToPdf();
        });
        $("#exportToExcelButton").click(function () {
            reportViewer.exportToExcel();
        });
        $("#exportToHtmlButton").click(function () {
            reportViewer.exportToHtml();
        });
        $("#exportToXpsButton").click(function () {
            reportViewer.exportToXps();
        });
        $("#firstPage").click(function () {
            reportViewer.firstPage();
        });
        $("#prevPage").click(function () {
            reportViewer.prevPage();
        });
        $("#nextPage").click(function () {
            reportViewer.nextPage();
        });
        $("#lastPage").click(function () {
            reportViewer.lastPage();
        });
    });
    function Initialize() {
        reportViewer = new PerpetuumSoft.Reporting.WebViewer.Client.ReportViewer("#ReportViewerElement");
        reportViewer.setServiceUrl("http://localhost:5555/WcfReportService.svc");
        reportViewer.reportName = "CustomersReport";
        reportViewer.documentInfoLoadedEvent = function (pages) {
            $("#pageCount").text(pages.length);
        };
        reportViewer.currentPageChangedEvent = function (pageNumber) {
            $("#currentPage").val(pageNumber);
        };
        reportViewer.errorEvent = function (errorModel) {
            switch (errorModel.errorType) {
                case PerpetuumSoft.Reporting.WebViewer.Client.ErrorType.communicationError:
                    alert("communicationError" + errorModel.error._error$1);
                    break;
                case PerpetuumSoft.Reporting.WebViewer.Client.ErrorType.clientError:
                    alert("clientError" + (errorModel.error).message);
                    break;
                case PerpetuumSoft.Reporting.WebViewer.Client.ErrorType.serverError:
                    alert("serverError" + (errorModel.error).message + (errorModel.error).getInformation());
                    break;
                default:
                    alert(errorModel.error.message);
                    break;
            }
        };
        reportViewer.renderDocument();
        reportViewer.setThumbnailsControl("#ssr_thumbnailContentPanel");
        reportViewer.setDocumentMapControl("#documentMapView");
    }
</script>
```

Now, you can run your web application. You will see the page with new design in the browser. It looks as you can see in the screenshot below:



If you have any questions regarding the integration with SharpShooter Reports.Web don't hesitate to contact us at support@perpetuumsoft.com.