Telerik Reporting

a step by step learning guide

telerik

Telerik Reporting

1 Introduction

About Telerik

Telerik Corporation was founded in 2002 and is a leading vendor for award winning ASP.NET and Windows user interface components. Telerik customers include Fortune 500 companies, education, government and non-profit organizations all over the world: Microsoft, United Nations, US Army, Harvard University, Intel, Siemens, Pfizer, Citigroup and NASA, just to name a few. Telerik staff has an impressive 11 MCSDs, 1 MCSE, 1MCDBA and 1 MCAD on its team of over 70 highly qualified professionals.

Who should read this?

You should read this course if:

- You have never used Telerik Reporting and want to learn what it is all about.
- You have used an earlier version of Telerik Reporting and want to become familiar with using the new features.
- You have used Telerik Reporting and want to make your knowledge more comprehensive.

What you need to have before you read this

Platform

Any of the following platforms may be used to run Telerik Reporting and exercises in this courseware, along with Microsoft .NET Framework 2.0:

- Windows 2000 Professional
- Windows 2000 Server
- Windows XP Home, SP2
- Windows XP Professional
- Windows 2003 Server
- Windows Vista

IDE: Visual Studio 2005/2008 in Standard, Professional, Team, Express editions.

Programming Languages: C# or VB.NET.

Also of course, Telerik Reporting must be installed. This will in turn install the AdventureWorks MSSQL database that is required for many of the labs in this courseware.

What you need to know before you read this

This courseware assumes you are familiar with either C# or VB.NET. You will need to be familiar with the Visual Studio IDE (VS 2005 was used to make the courseware) and know your way around this environment. You should be able to navigate the basic functional areas of the IDE (e.g. Solution Explorer, Properties, events, design/source for web pages, etc.) You should be able to run and debug Windows and Web applications.

How this tutorial is organized

Getting Started with Telerik Reporting

In this section you will use best practices to construct a simple "Hello World" single page report without a database, then create a simple database listing of products. Finally you learn how to display your report in a ReportViewer control.

Telerik Reporting Design Environment

You will take a tour of the report design environment, the report sections that house content and briefly sample the report items that are placed within the report sections.

Connecting to Data

This section defines the underlying types of data that can be bound to a report. You will first learn how to bind very simple array type data, then how to bind database data manually and programmatically.

Expressions

This section reviews the types of expressions available, how to create your own user defined functions and how expressions are used within the report

Grouping

This lesson demonstrates the basic usage of groups and grouping related properties, how to create groups using the designer and how to create groups using code only.

Filtering

This section explores how filter rules work when they are applied to report data and report groups. You will create a filter with multiple rules and use it to narrow down a range of report data. You will recreate this same filter in code. This section also looks at performance considerations for filtering using the mechanisms supplied by Telerik Reporting in contrast with using the database engine on a server.

Sorting

This section explores how sorting works against report detail data and how you can also sort groups. You will learn how to apply sorting at design time and programmatically.

Parameterized Reports

In this section you will learn how parameters work with expressions to create dynamic, powerful and maintainable reports. Parameters creation is demonstrated at both design-time and run-time. You will learn how to automatically include simple and cascading parameters in your user interface. You will create a report viewing Windows application that automatically displays parameter prompts.

SubReports

This section demonstrates how a SubReport item is used to display one report within another report, allowing you to compose complex reports from multiple report sources. You will learn how to create aggregate reports from disparate report sources, master-detail reports and how to hide SubReports based on expression criteria.

Report Appearance and Styling

This section explores the array of choices in Telerik Reporting for altering the appearance of your report, including styles, style rules and conditional formatting. You will learn how to define a style sheet, export a stylesheet and to re-use a stylesheet in other reports.

Conditional Formatting

This section demonstrates how to perform common useful tasks such as displaying alternating row colors and highlighting report items based on data conditions using conditional formatting.

Reports at Runtime

This lesson shows how to create report sections and items at run-time. You will also become familiar with the run-time specifics of the TextBox, Shape and PictureBox report items.

Events

In this section you will be introduced to the report life cycle - a critical part of understanding how Telerik Reports function. The report, section and item events are shown in the context of the report life cycle.

Paper Size

This section will show you how to print sheets of labels from a datasource, how to print data that displays in multiple columns, and how to create reports that display in custom paper dimensions.

Handling Report Output

In this section you will become familiar with two Telerik Reporting controls used for viewing reports within Winforms and Web applications. You will also export your reports programmatically to various file formats on disk, including image and PDF.

2 Acknowledgements

Telerik Reporting

Author:

Noel Rice

Design:

Matt Kurvin

Acknowledgement:

Ivo Nedkov

Vassil Terziev

Nikolay Dobrev

Lino Tadros

Todd Anglin

Vassil Petev

Svetozar Georgiev

Production:

Falafel Software Inc.

www.falafel.com (http://www.falafel.com/)

3 Getting Started with Telerik Reporting

Objectives

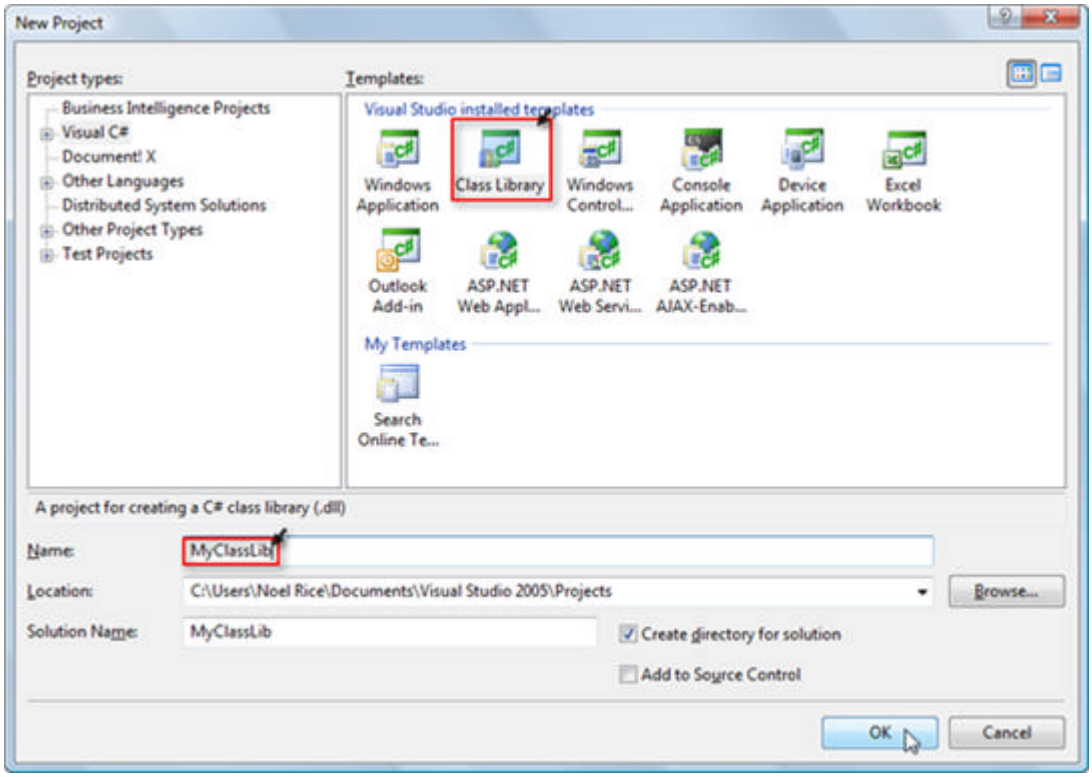
- Learn best practices to make reports reusable in Windows and Web applications.
- Use the Report designer user interface manually to produce a very simple "Hello World" output.
- Use the view mode tabs **Design**, **Preview** and **HTML Preview**.
- Use the Report Wizard to build a simple data listing.
- Display a report in a Windows forms viewer.

Lab: Creating Simple Output

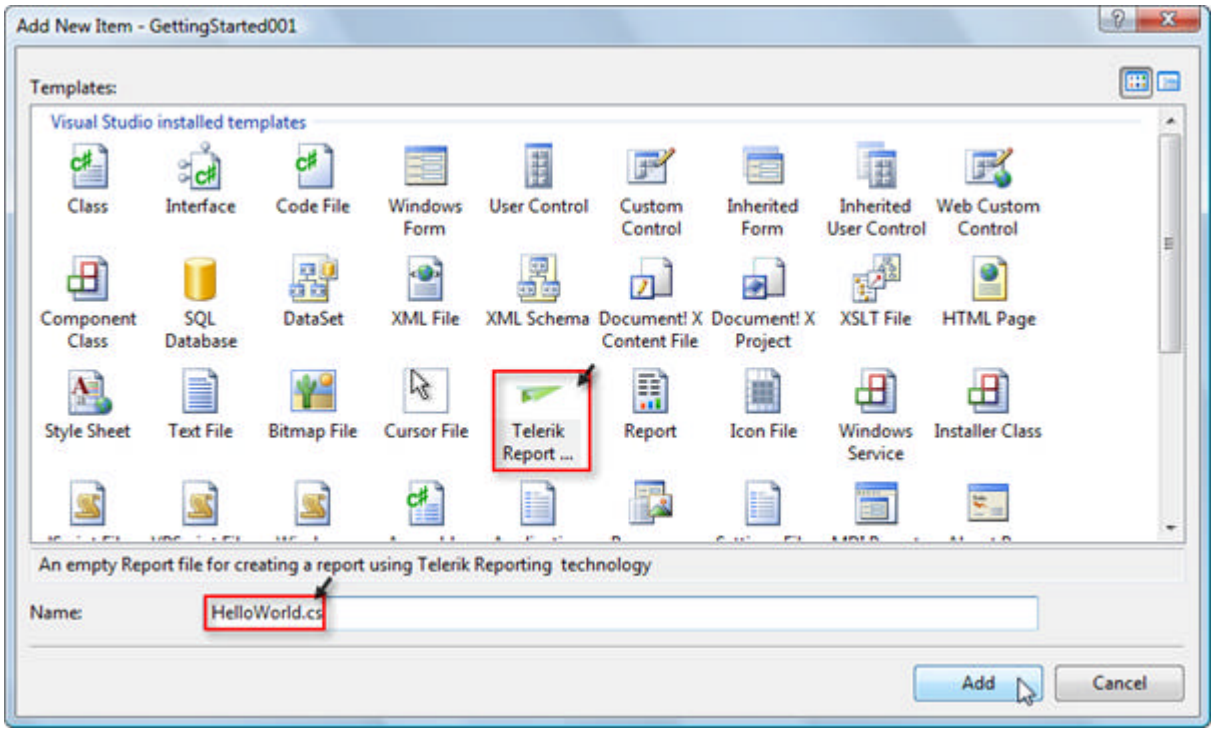
Before we begin, a word about best practices. Telerik reports can be created directly within windows or web projects, but in the course of this training, we will ***always*** store our reports in a class library. In this way, you can reference your class library from multiple projects, in either Windows or web applications. You can have any number of Windows or Web viewing applications that re-use the same class library.

To create even simple reports you start by creating a class library to contain your report. In this walk-through of a "Hello World" report, you will put some basic text on a report header, footer and detail. All the text in the report will be static.

1. Select **File | New | Project** from the Visual Studio File menu. Select the **Class Library** project, give it a name and location. Click the **OK** button to close the dialog.



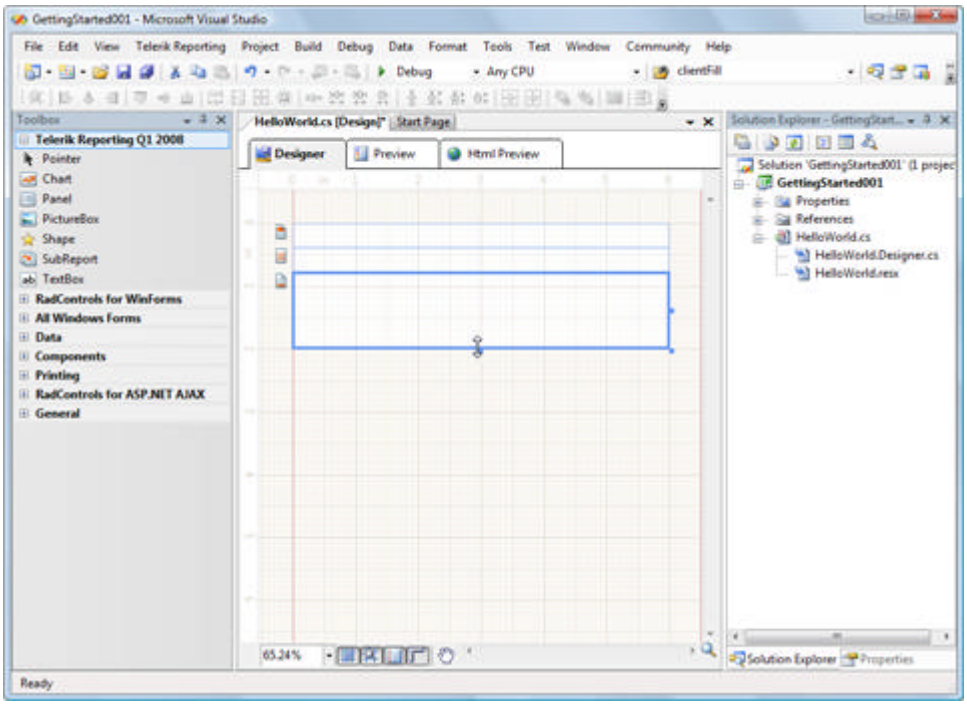
2. Right-click the project context menu and select **Add | New Item | Telerik Report.** Enter a name for the report class and click the **Add** button to close the dialog. In this example we name the report class file "HelloWorld.cs".



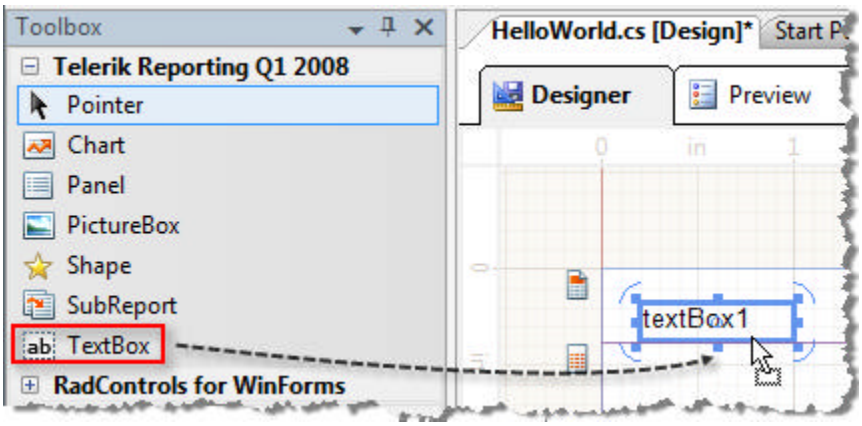
3. The Telerik Report Wizard will appear automatically to help you select data and design your report quickly. In this case, click the **Cancel** button so we can create a "Hello World" report manually with static data.
4. Click **Yes** to confirm the "Are you sure you want to cancel this wizard?" dialog.

> You can overwrite your existing report at any time from the Report Wizard option of the Visual Studio Telerik Reporting menu.

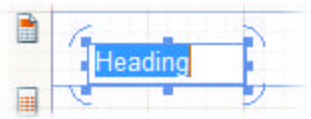
5. The Telerik Report designer displays sections for a basic report with a page header, detail and footer. The Toolbox window displays the controls that can be dragged to the report design surface. Later we will dig into more specifics of the report designer and the kind of sections that can be included in the report, and how each type of Telerik Reporting component behaves.
6. Use the mouse to resize the header, detail and footer sections to make each section smaller. The mouse cursor will display drag handles as you pass it over the very top of each section heading and also along the bottom border of the report footer.



8. Drag a **TextBox** into the **PageHeaderSection** of the report designer.



10. Double-click the TextBox and edit the text to read "Heading". **Note**: This step assigns the TextBox **Value** property.



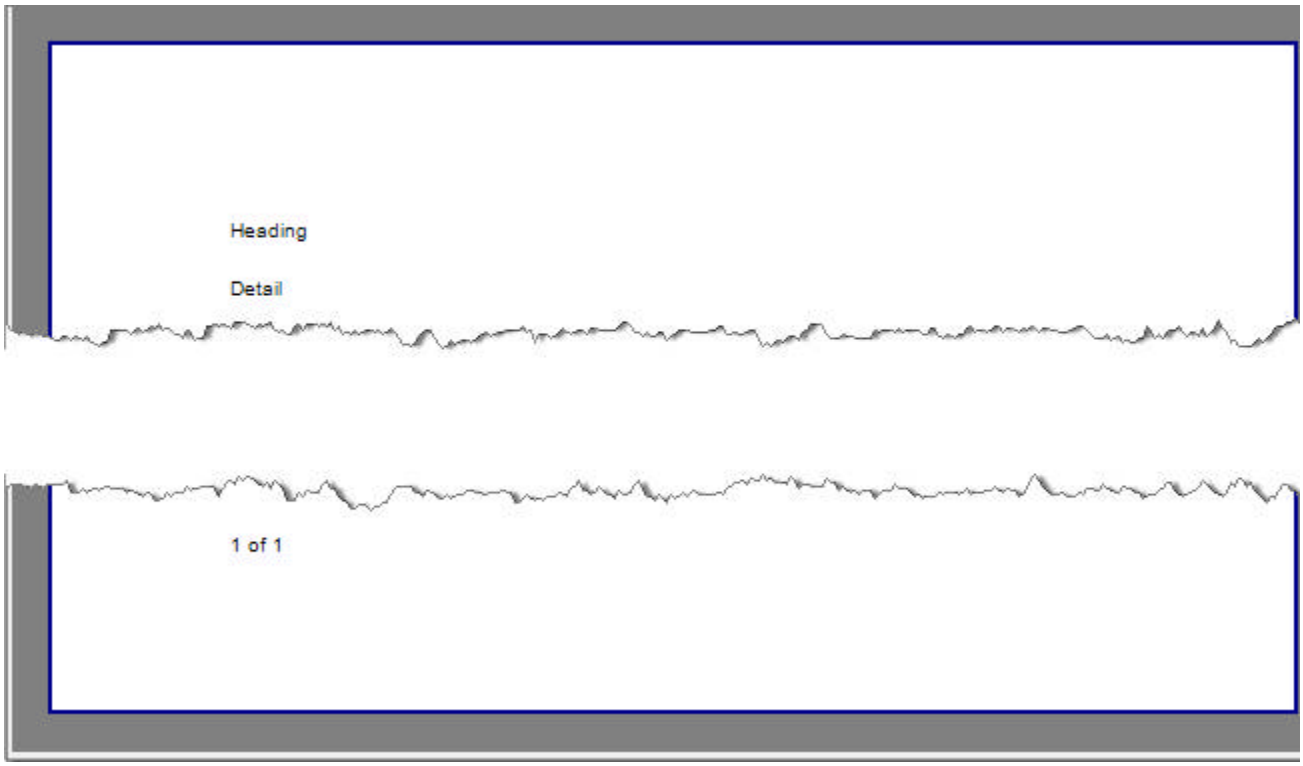
12. Drop a TextBox component in the **Detail** section and change the text to "Detail".
13. Drop a TextBox component in the **Footer** section and change the text to

    "=PageNumber + " of " + PageCount".

    Note: We will talk later about how expressions are created, but for now just copy the expression into the TextBox.

14. Click the **Preview** tab at the top of the Report Designer to display the report.

    This is how the report will look in a **ReportPreview** control. Notice the layout of the report with the heading and detail toward the top of the report, and the footer at the bottom of the report.

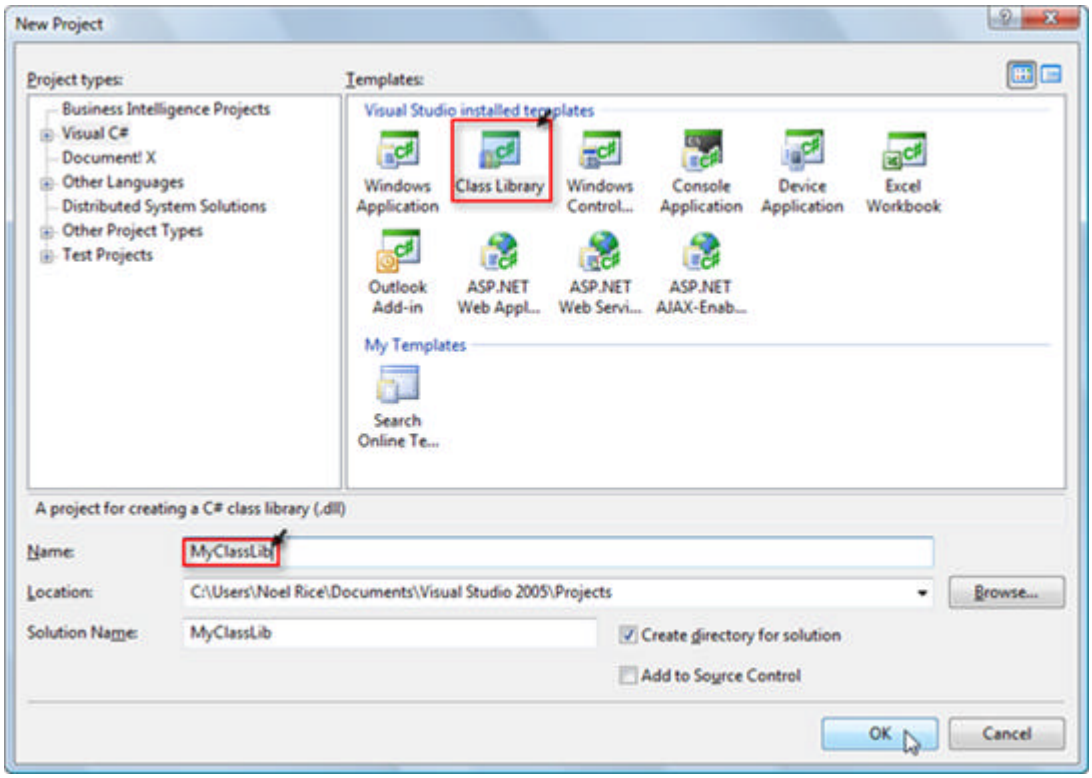


16. Now click on the **HTML Preview** tab to see how the report output will look in a browser. Notice that the HTML rendering does differ from the Winforms preview.

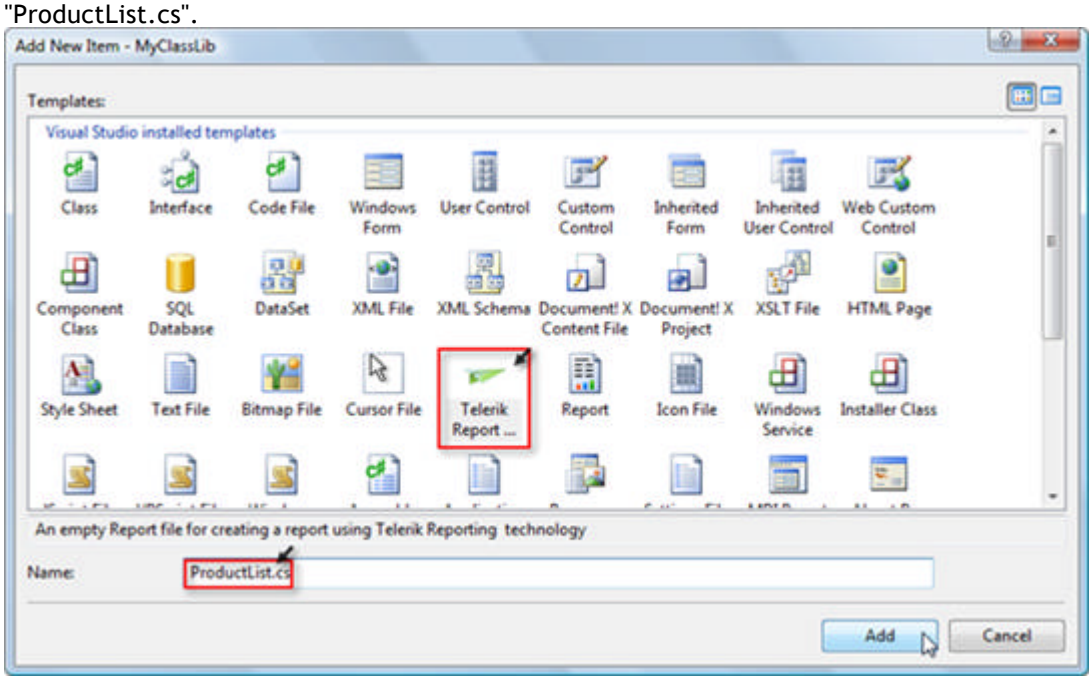
Lab: Creating a Simple Database Report

Typical business reports are going to consume database data. The steps here will show you how to produce a simple, single-level database listing. This is the kind of report you might expect when listing phone numbers, employee names, or as in this case, a list of products.

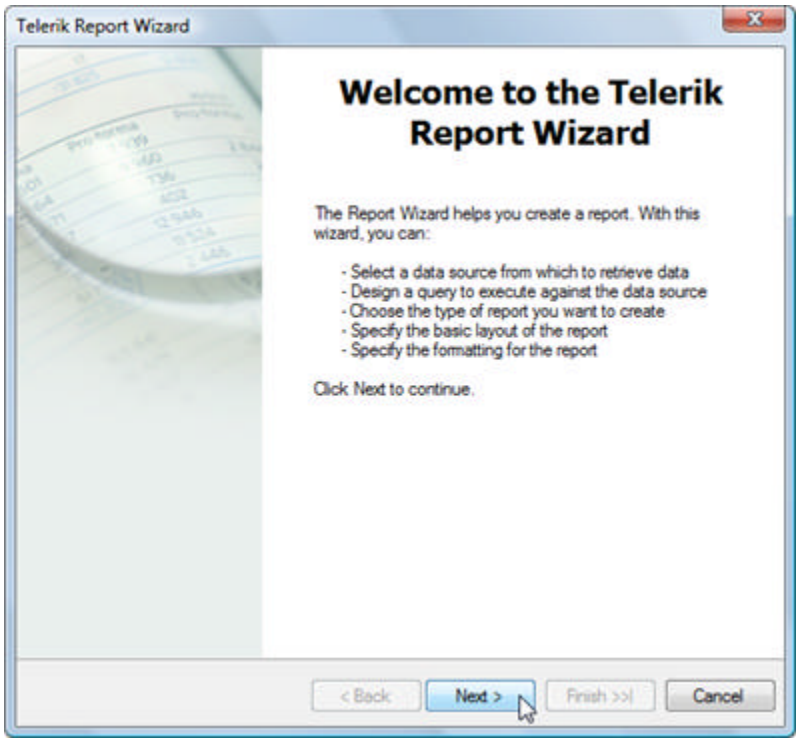
1. Select **File | New | Project** from the Visual Studio File menu. Select the **Class Library** project, give the class library a name and location. Click the **OK** button to close the dialog.



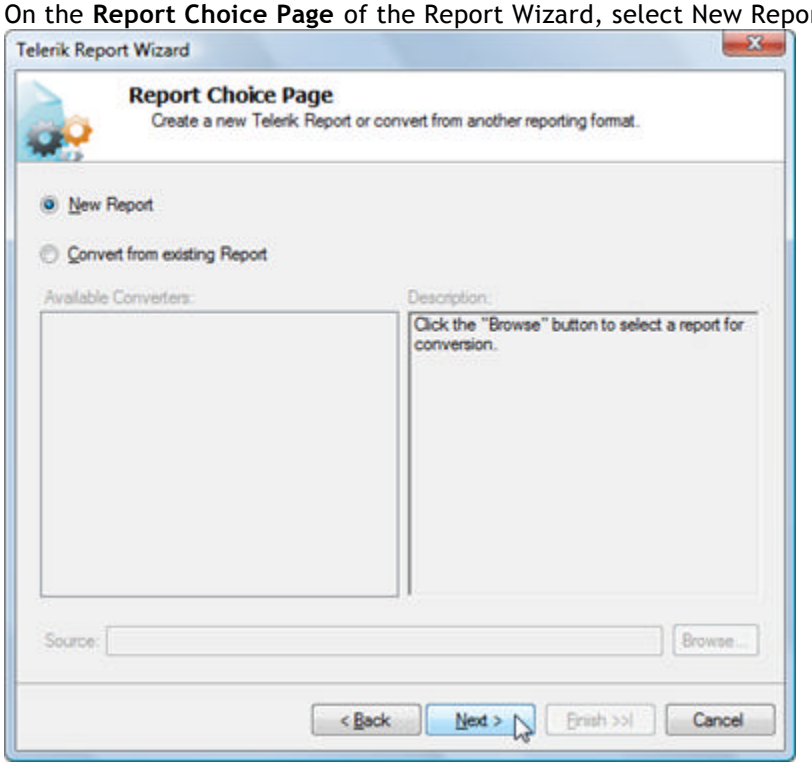
2. Right-click the project context menu and select **Add | New Item | Telerik Report.** Enter a name for the report class and click the **Add** button to close the dialog. In this example we name the report class file "ProductList.cs".



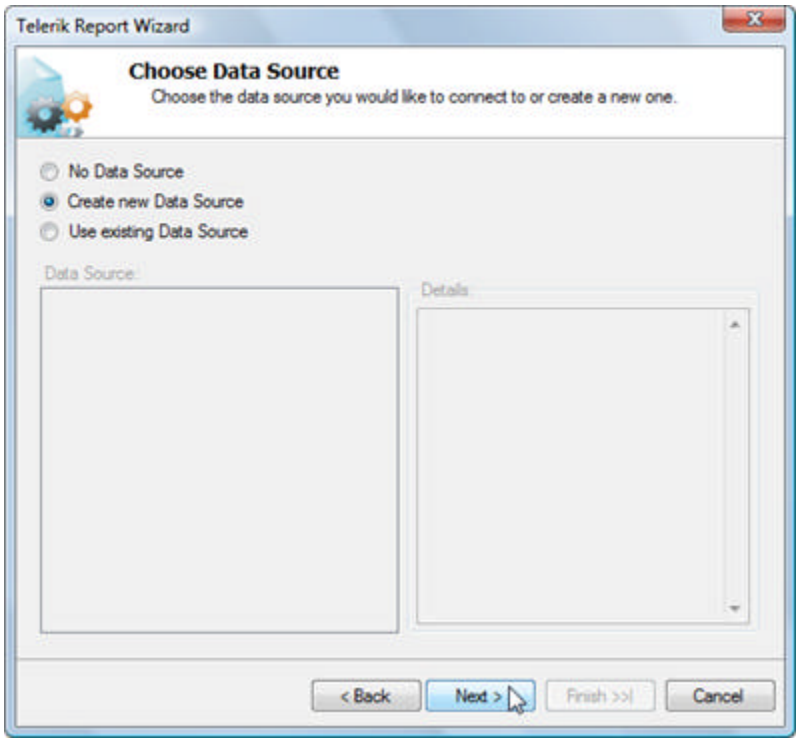
3. The Telerik Report Wizard will appear automatically to help you select data and design your report quickly. Once you have the basic report layout you can go back to edit the report, add data columns and tweak the format. Click the **Next** button.



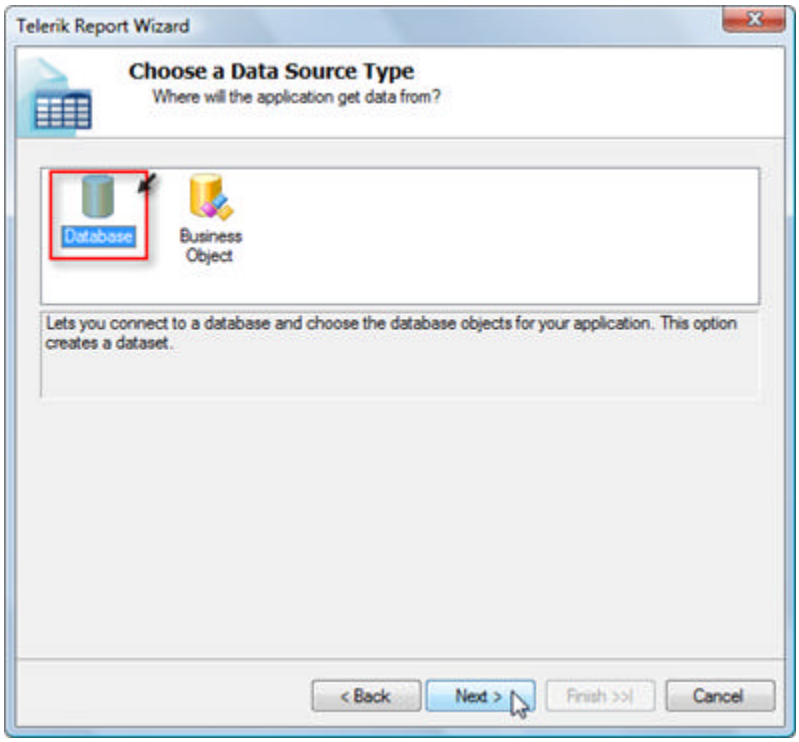
4. On the **Report Choice Page** of the Report Wizard, select New Report and click the **Next** button.



5. On the **Choose Data Source page** of the Report Wizard, select **New Data Source** and click the **Next** button.



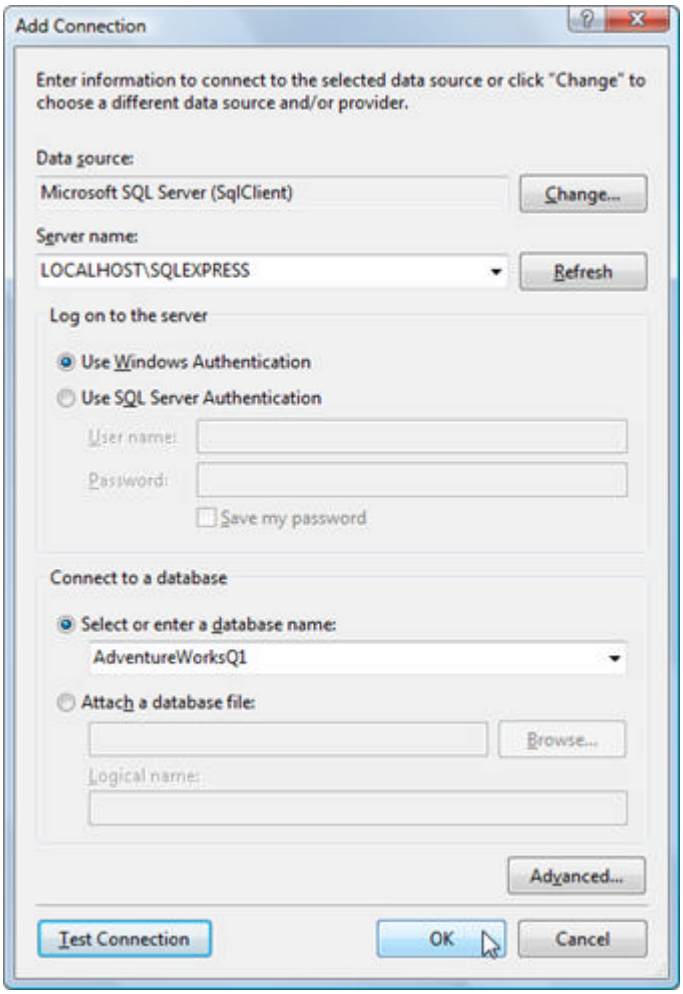
6. On the **Choose a Data Source Type** page of the Report Wizard select the **Database** icon and click the **Next** button.



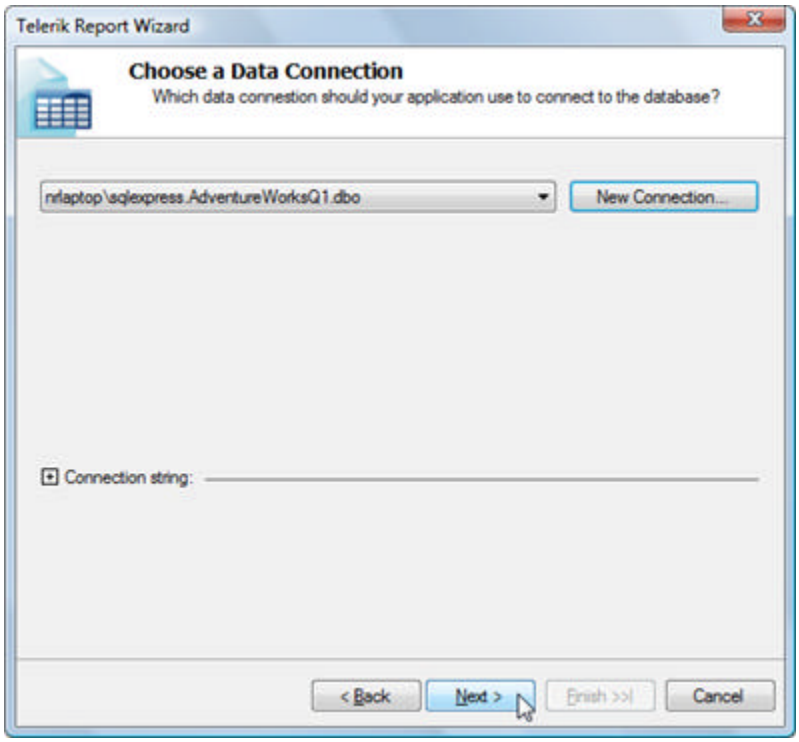
7. On the **Choose a Data Connection** page of the Reports Wizard click the **New Connection...** button.
8. In the **Add Connection** dialog:
    1. Make sure the Data source is "Microsoft SQL Server".

2. Set the Server name to "LOCALHOST\SQLEXPRESS".
3. Verify that for "Log on to the server" has "Use Windows Authentication" selected.
4. For "Connect to a database" under "Select or enter a database name" enter ***"AdventureWorks"**** . Click the **OK** button to close the Add Connection dialog.

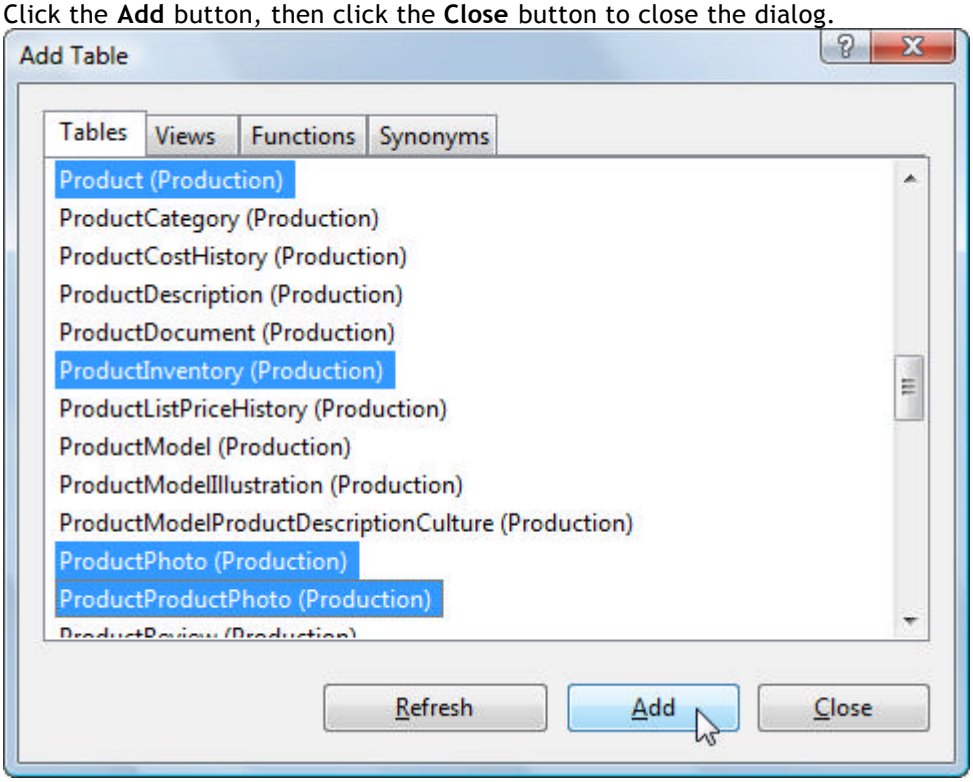
> ⚠ * The AdventureWorks database is installed along with Telerik Reporting and may change with different versions of Telerik Reporting. For example, the database name at the time of this writing was "AdventureWorksQ1". You can select the current name of the database from the drop down list.



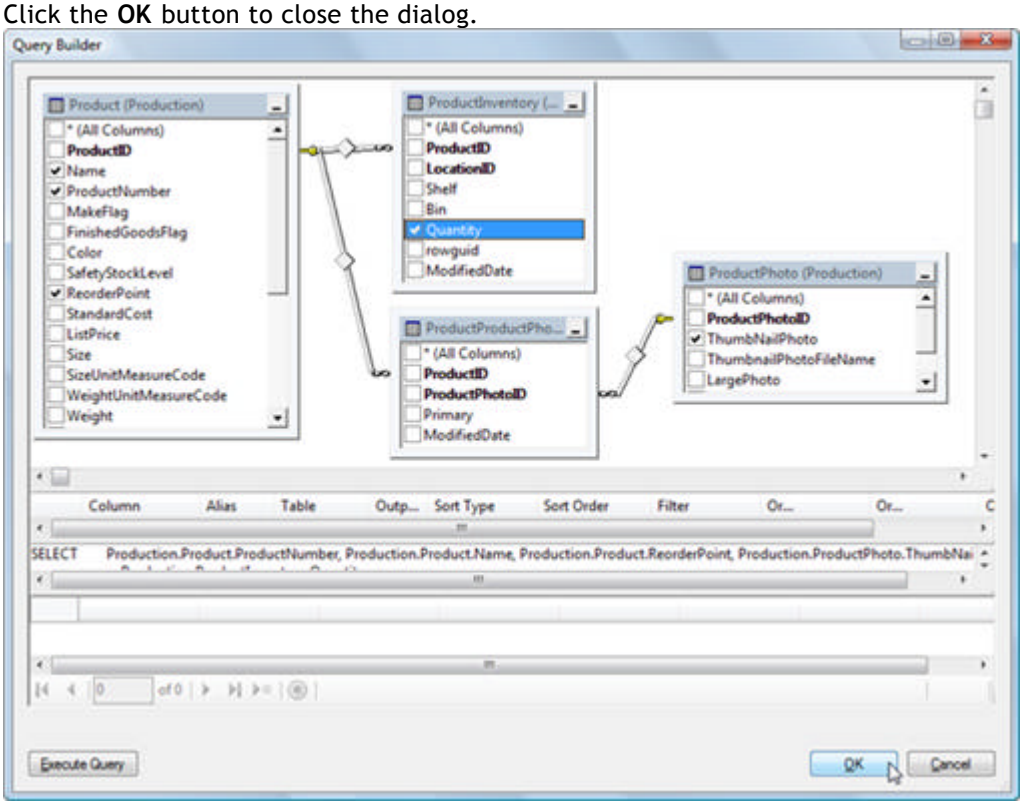
9. On the **Choose a Data Connection** page of the Report Wizard, verify that your new database connection for AdventureWorks is selected from the drop down list. **Note**: The machine name will be the name of the machine you are developing on, i.e. <my machine name>sqlexpress.AdventureWorks.dbo. Click the **Next** button.



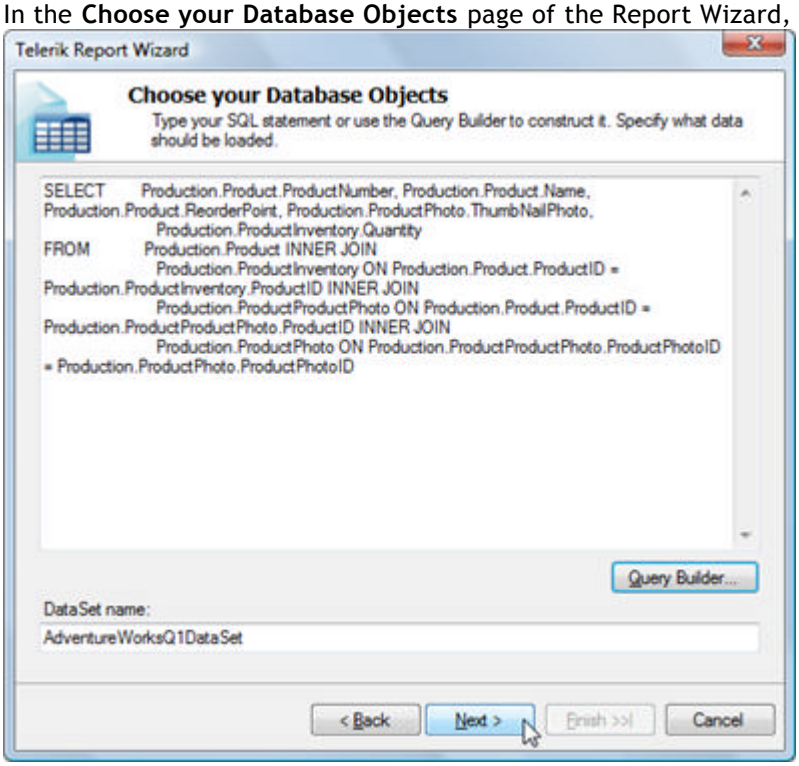
10. In the **Choose Your Database Objects** page of the Report Wizard:
    1. Click the **Query Builder** button.
    2. In the **Add Table** dialog select "Product", "ProductInventory", "ProductPhoto" and "ProductProductPhoto" from the list of tables. **Note**: You can hold down the control key and click each of the tables to select them all at one time.
    3. Click the **Add** button, then click the **Close** button to close the dialog.



4. On the **Query Builder** select the fields shown in the figure below. In the Product table select "Name", "ProductNumber" and "ReorderPoint". In ProductPhoto select the "ThumbNailPhoto" column. In the ProductInventory table select the "Quantity" column.
5. Click the **OK** button to close the dialog.

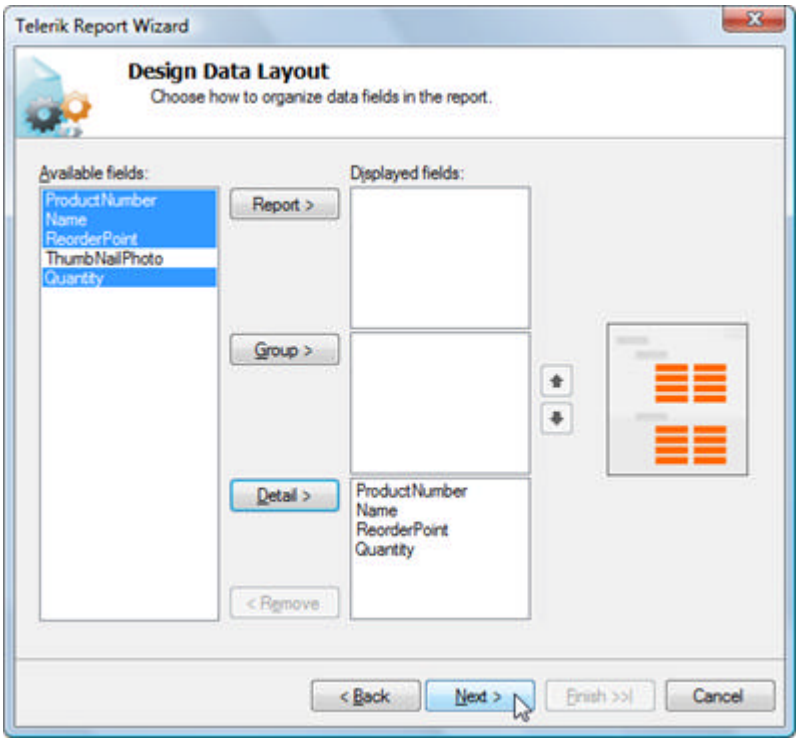


6. In the **Choose your Database Objects** page of the Report Wizard, click the **Next** button.

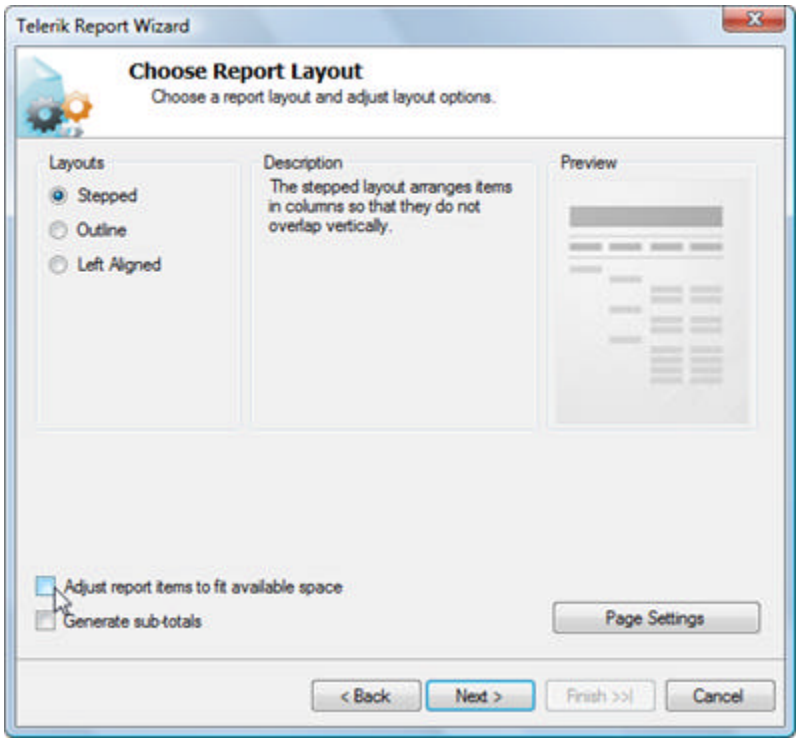


Now that the data is configured, the remainder of the Report Wizard pages are concerned with report content layout. These next steps will include selecting the columns to display, making column arrangement choices, and choosing a report style that will set the overall look and feel for the report.

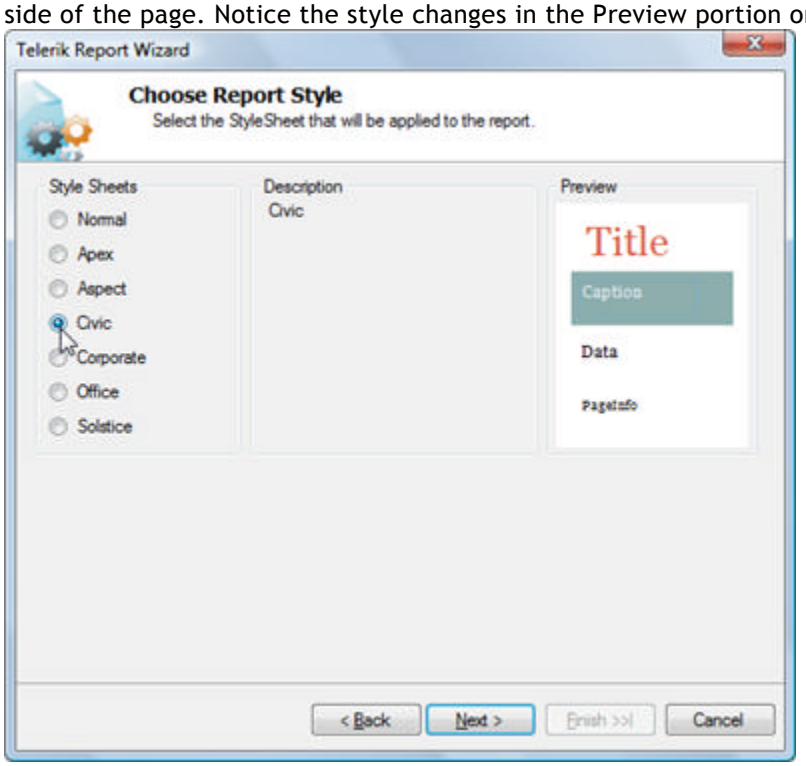
11. Select **Standard** from the **Select the Report Type** page of the Report Wizard.
12. The **Design Data Layout** page of the Report Wizard allows you to assign database fields to sections of the report. The Report Wizard automatically places and formats the database fields in the appropriate report sections.
    1. Select from the **Available Fields** list on the left side of the page.
    2. Select "Name", "ProductNumber", "ReorderPoint" and "Quantity".
    3. After selecting each field, click the **Detail** button to add those columns to the detail listing of the report.
    4. Click the **Next** button.



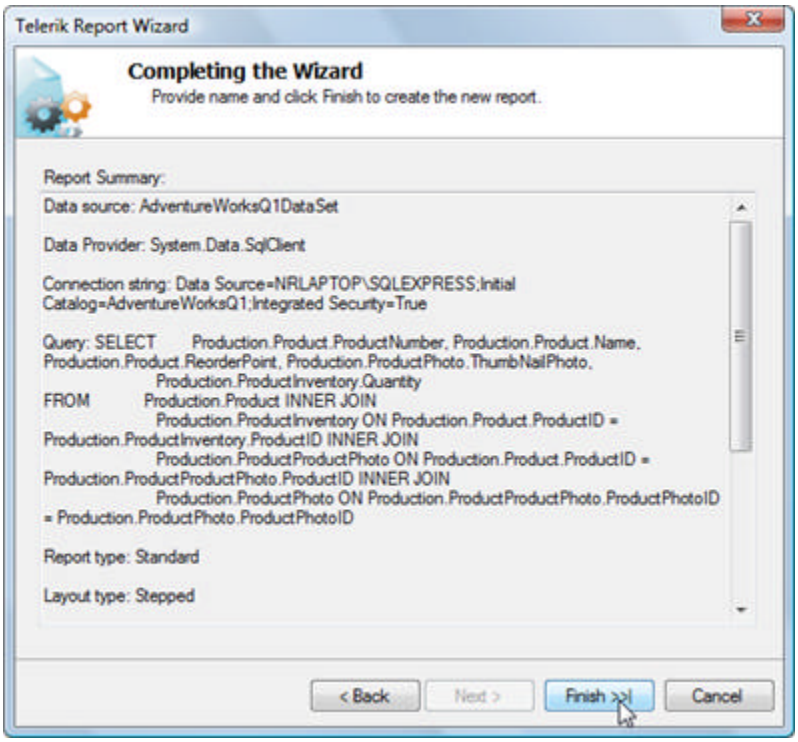
13. On the **Choose Report Layout** page of the Report Wizard unselect the **Adjust report items to fit available space** checkbox. Click the **Next** button.



14. On the **Choose Report Style** page of the Report Wizard select "Civic" from the **Style Sheets** list on the left side of the page. Notice the style changes in the Preview portion on the right side of the page.



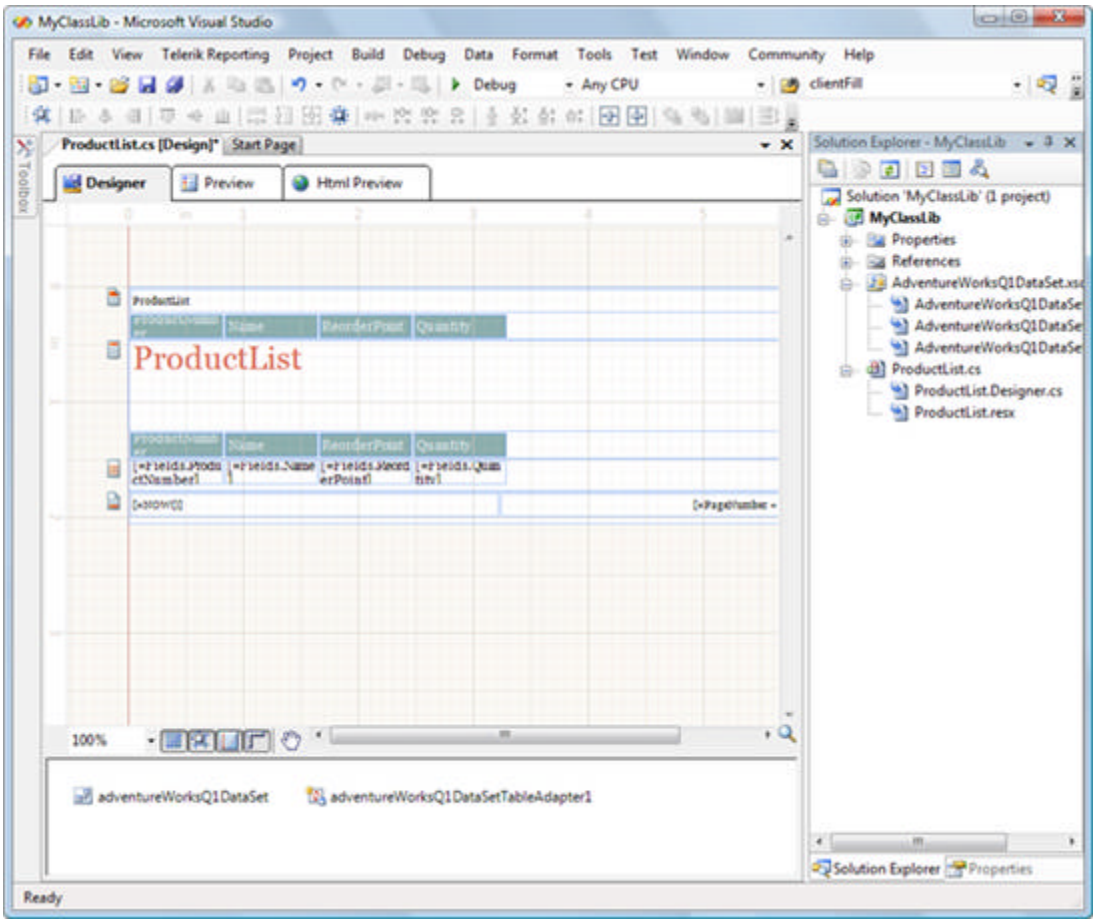
15. On the Completing the Wizard page of the Report Wizard you can review the settings for the report and click the **Finish** button.



The initial report layout appears in the design view in Visual Studio.

Notice that the wizard has automatically provided:

- Data set adapter and data set components.
- Data bound fields in the detail section of the report.
- Styled page and column titles.
- A page footer with standard date and page number output.



16. Click the **Preview** tab of the designer to view your report.

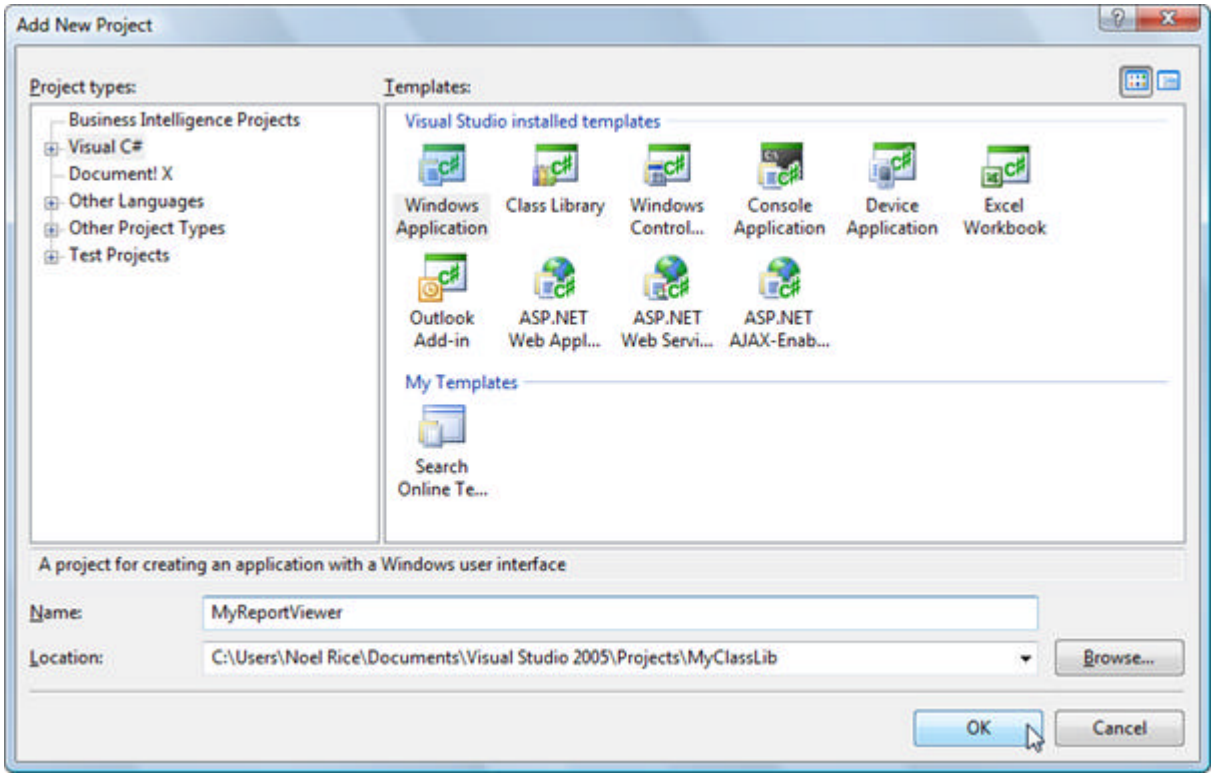
ProductList

ProductNumber	Name	ReorderPoint	Quantity
AR-5381	Adjustable Race	750	408
AR-5381	Adjustable Race	750	324
AR-5381	Adjustable Race	750	353
BA-8327	Bearing Ball	750	427
BA-8327	Bearing Ball	750	318
BA-8327	Bearing Ball	750	364
BE-2349	BB Ball Bearing	600	585
BE-2349	BB Ball Bearing	600	443
BE-2349	BB Ball Bearing	600	324

Lab: Displaying Reports in a Viewer

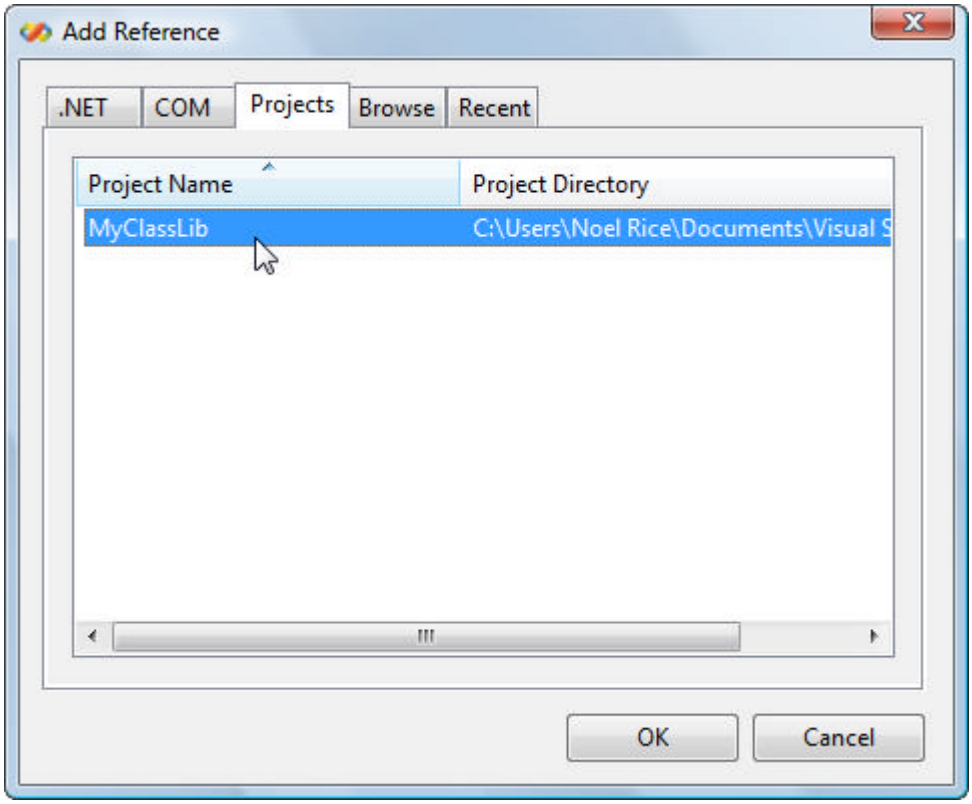
Now that we have a report defined within a class library we can consume it in a Windows or Web viewer application. The steps here will walk you through creating a Windows report viewing application that displays the Product Listing report.

1. Select **File | New | Project** from the Visual Studio File menu. Select the **Windows Application** project, give it a name and location. Click the **OK** button to close the dialog.

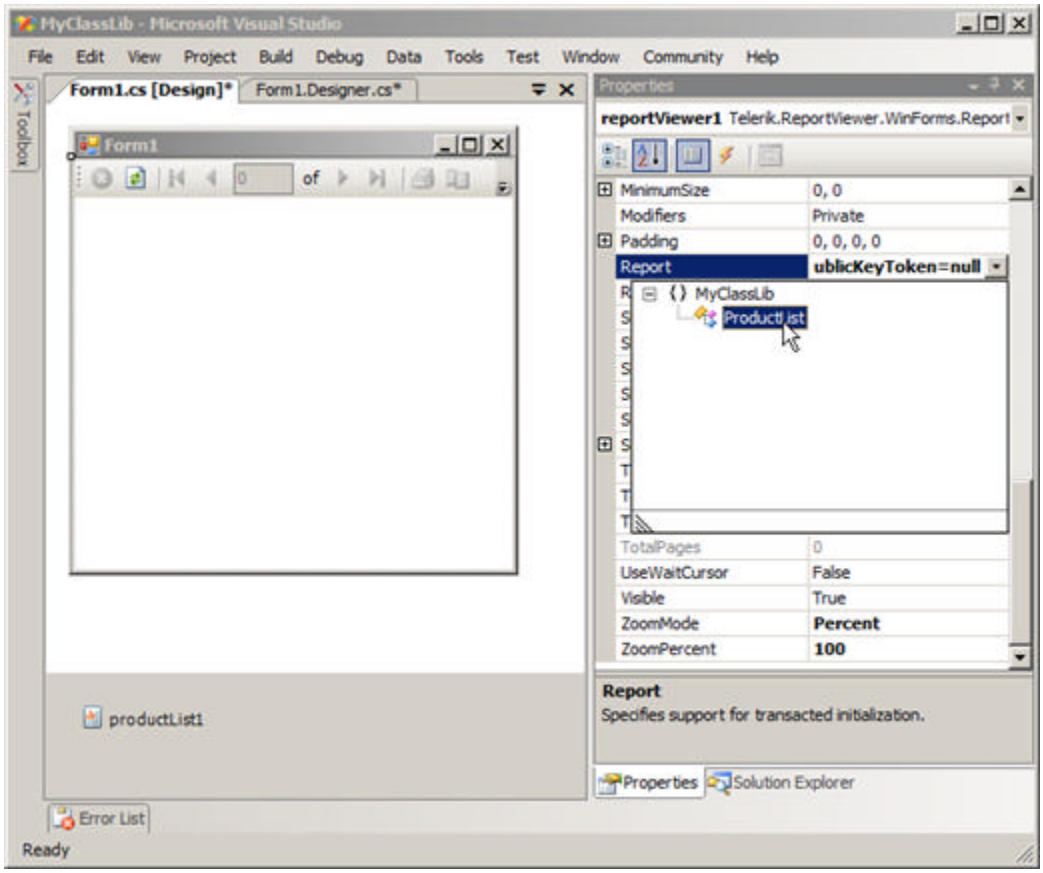


2. In the Solution Explorer, right-click the project References node and select **Add Reference** from the context menu. Select the reference for the class library containing your report from the list on the **Projects** tab.

   Note: If the class library resides in another solution you will need to use the **Browse** tab and navigate to the assembly containing the report.



3. From the ToolBox drag a **ReportViewer** control to the Windows form.
4. Set the **Dock** property to **Fill** so that the viewer takes up the entire window area.
5. Set the **Report** property from the drop down list. This will be the report that was defined in the class library.



> Once the report is selected, notice that a component representing the report appears in the tray below Windows form designer. The component allows you to access the properties of the report in your report viewing application, providing access to report parameters, styles and page settings.

6. Add a call to the report viewers RefreshReport() method, just after the call to InitializeComponent() in the form constructor.

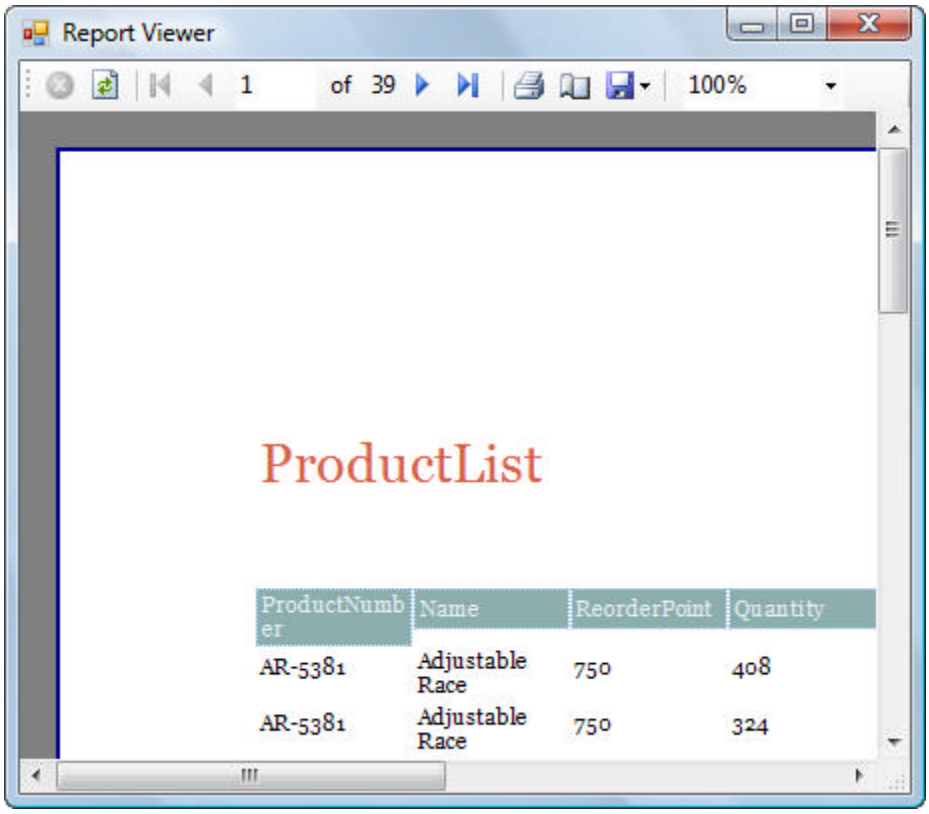
[C#] Calling RefreshReport()

```
public Form1()
{
   InitializeComponent();
   reportViewer1.RefreshReport();
}
```

[VB] Calling RefreshReport()

```
Public Sub New()
 InitializeComponent()
 reportViewer1.RefreshReport()
End Sub
```

7. Right-click the Windows report viewing application and select **Set as Startup Project** from the context menu.
8. Press **F5** to run the report.



Summary

In this section you used best practices when constructing your reporting solutions for best code reuse. You created a simple "Hello World" single page report without a database and also you created a simple database listing of products. Finally you displayed your report in a Windows form viewer.

4 Telerik Reporting Design Environment

Objectives

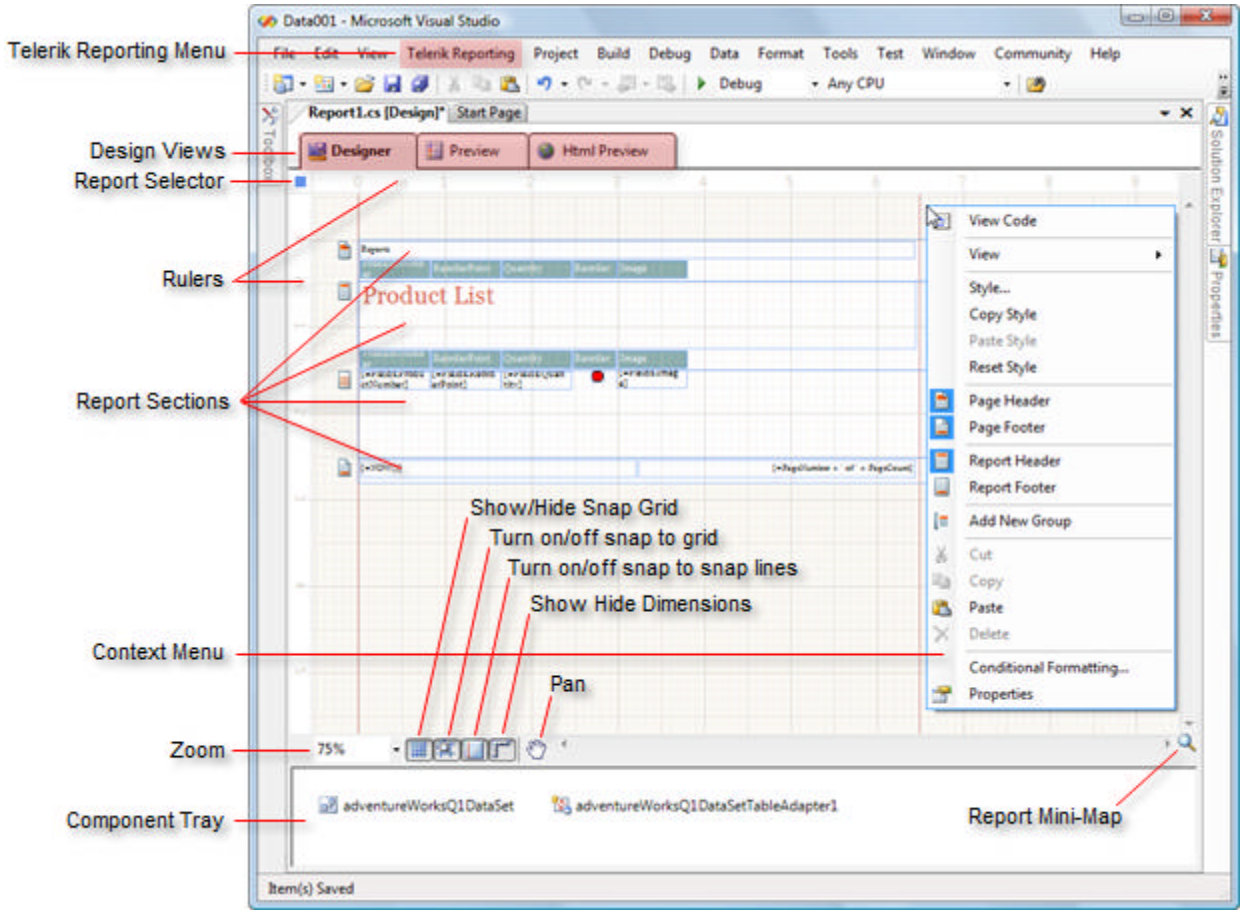
- Become familiar with the Report Designer. Learn how each section of the report design surface contributes to the overall report.
- Learn about the context menus and how they work with each portion of the report design surface.
- Use the Telerik Reporting Visual Studio menu **Report Explorer**, **Data Explorer** and **Report Wizard** options.
- Learn the purpose of each reporting item in ToolBox window.

A Tour of the Report Designer

The Telerik Reporting design environment comes with all the tools for building complex reports quickly. Most tasks can be performed directly in the design surface; this surface simulates graph paper and represents the actual layout of report elements as they would appear on a printed sheet of paper. There are no bulky section dividers or other elements that clog the designer.

Here is a high level break out of the designer areas:

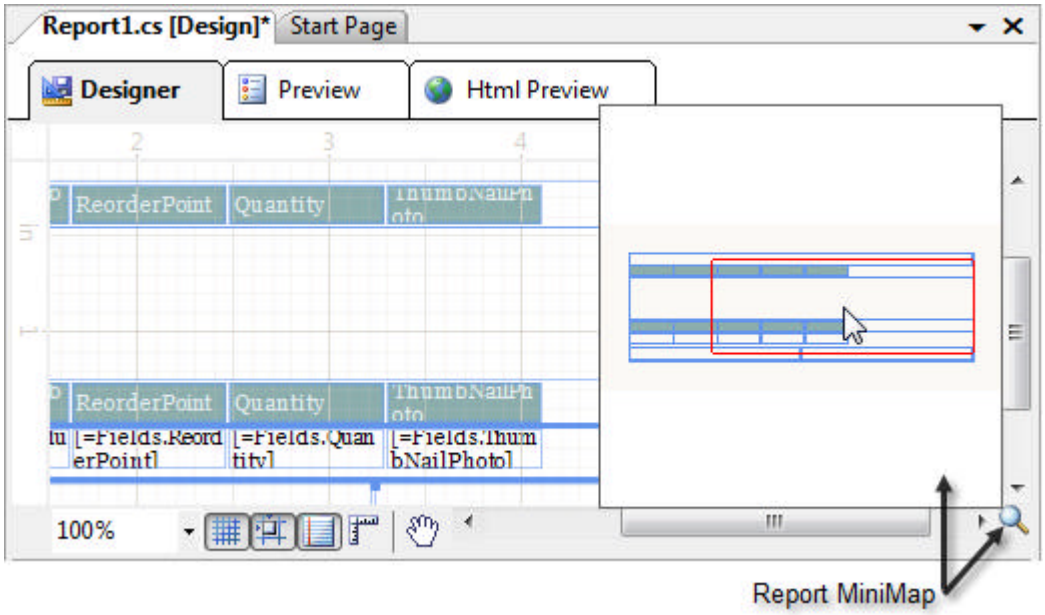
- The **Telerik Reporting Menu** holds options to display a **Report Explorer**, **Data Explorer** and to invoke the **Report Wizard**. **Note:** You can also reach the Data Explorer and Report Explorer by right-clicking next to the design area and selecting **View** from the context menu.
- The view mode buttons **Designer**, **Preview** and **HTML Preview** appear above the report design surface and control the overall behavior of the designer.
- The report design surface is the largest area containing report sections that make up the report.
- Below the report design surface, is a set of tools that allows you easier access and better control over the design environment. The **Zoom** drop down lets you enter a percentage or pick a predefined zoom. The next two tools let you toggle the snap-to-grid visibility and functionality. The **Snap to Snap Lines** tool when enabled lets you easily align a report item with other items on the design surface. The **Dimensions** tool displays report item measurements from the item to the edge of the report. The **Pan** tool when enabled lets you drag the report around the design surface - this can be handy when the report exceeds the easily viewable dimensions of the Visual Studio IDE. The **Report Mini-map** displays a thumbnail image of the report.
- Below the report design surface, the component tray may hold one or more data access or group components.



The elements within the report design surface are:

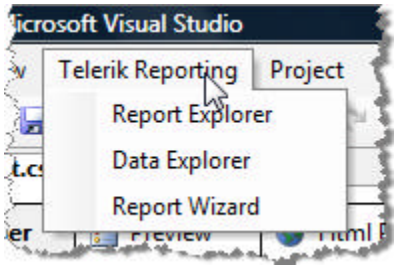
- **Report selector button:** Located in the upper left hand of the report designer. Clicking this button makes the report active in the **Properties** window.
- **Rulers:** on the top and left side of the designer provide a point of reference to the report layout. Click in the ruler area to display a line vertically or horizontally across the report. You may also drag in the ruler area to select an area vertically or horizontally across the entire report. This can be particularly useful when you want to select everything in a particular column or set of columns vertically.
- **Report Sections**: The high level report design consists of report sections for the report header, report footer, page header, page footer, detail, group header and group footer. You can select sections using the icon to the left of each section. Each section can be resized by dragging the sizing grips at the top of each section and collapsed using the button located to the left of each section. Most sections except the detail can be deleted by selecting the section and hitting the delete key.
- **Context Menu**: This menu will conditionally display contents depending on the area that was right-clicked.

Here is another view of the Report MiniMap showing how you can drag a box representing the visual area of the report:



Using the Telerik Reporting Menu

The **Telerik Reporting menu** can be accessed whenever an area of the report designer is selected. From the menu you can invoke the Report Explorer, Data Explorer and the Report Wizard.



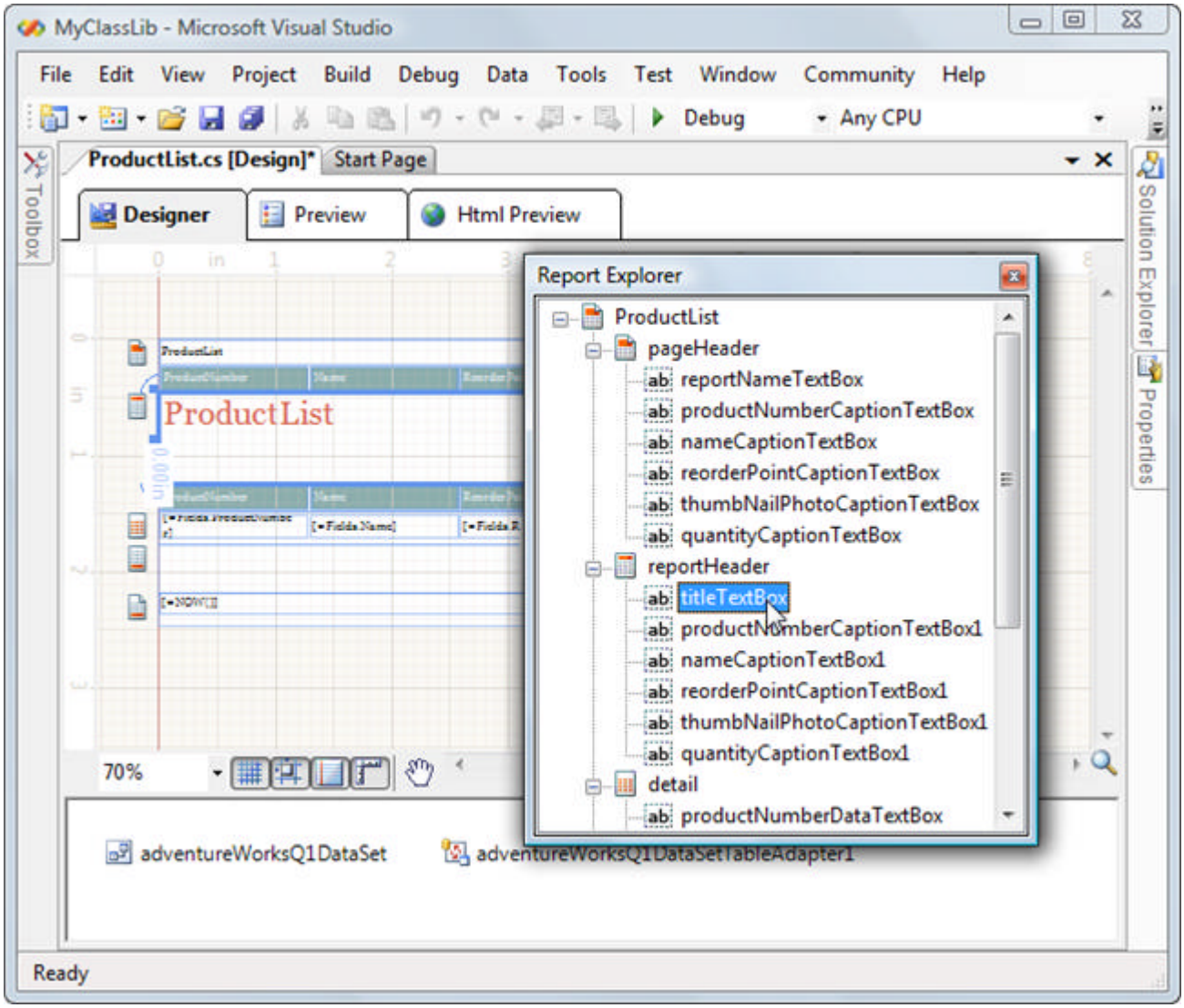
Report Explorer

The **Report Explorer** is an aid to navigating report elements. The Report Explorer allows you to see the structure of the report and to select any item in the report. The Report Explorer can be especially handy for complex reports where components may be nested and where it would be difficult to select an item with the mouse.

The **Report Explorer** can also be accessed from the context menu **View | Report Explorer** item when right-clicking the area next to the report design surface.

In Visual Studio, open the report project you built in the "Creating a Simple Database Report" tutorial. If you don't see the Telerik Reporting menu, click on the report design surface and it will appear. The Report Explorer shows a tree representation of the sections within the ProductList report class (i.e. page header, report header, etc.) and the report items within each section.

Clicking on report sections and items places focus on the corresponding entry. Try clicking the "detail" entry in the Report Explorer. Notice that the section title bar gets focus as you select it in the Report Explorer. Also try selecting report items.

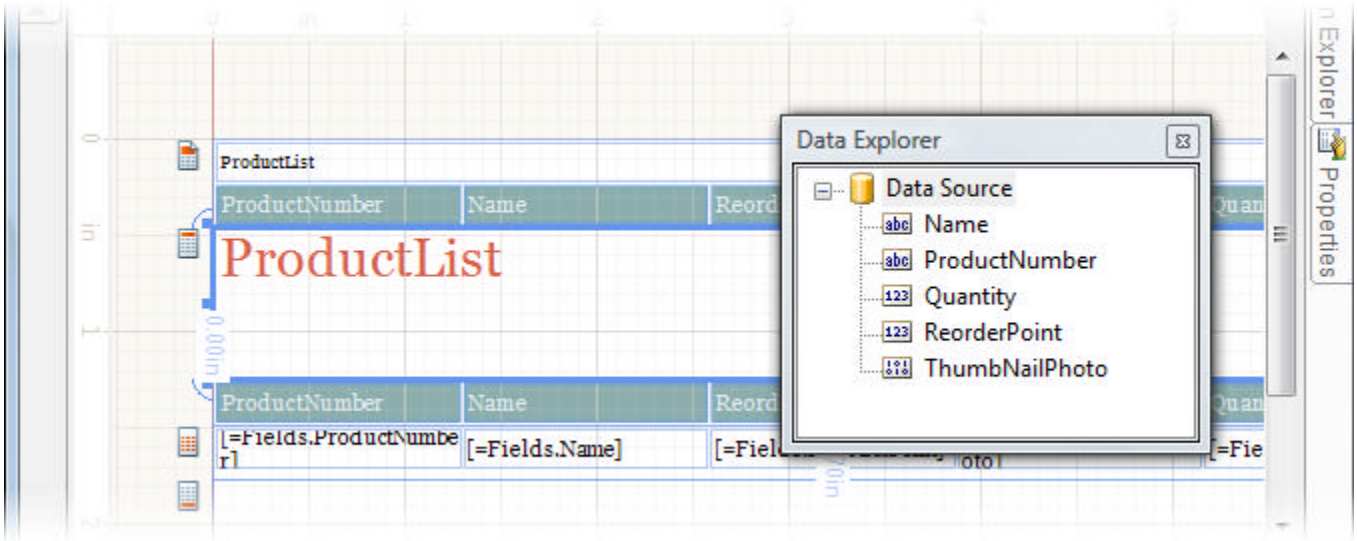


Data Explorer

The **Data Explorer** window lists the data source columns defined for your report. You can drag column names from the Data Explorer to the report design surface. The report designer automatically creates appropriate report items based on the column data type, e.g. if the column has character data a **TextBox** report item will be created, if the column has image data then a PictureBox report item will be created.

Using the "Creating a Simple Database Report" tutorial project, open the Data Explorer. Drag the "Name" column to the Detail section of the report designer. Notice that this action automatically creates a TextBox report item. Now try dragging the ThumbNailPhoto column to the Detail section of the report. This time a PictureBox report item is created.

Double-clicking a Data Explorer column name will create a report item in the currently selected report section. Try selecting the PageHeader, then double-click the "Name" column.

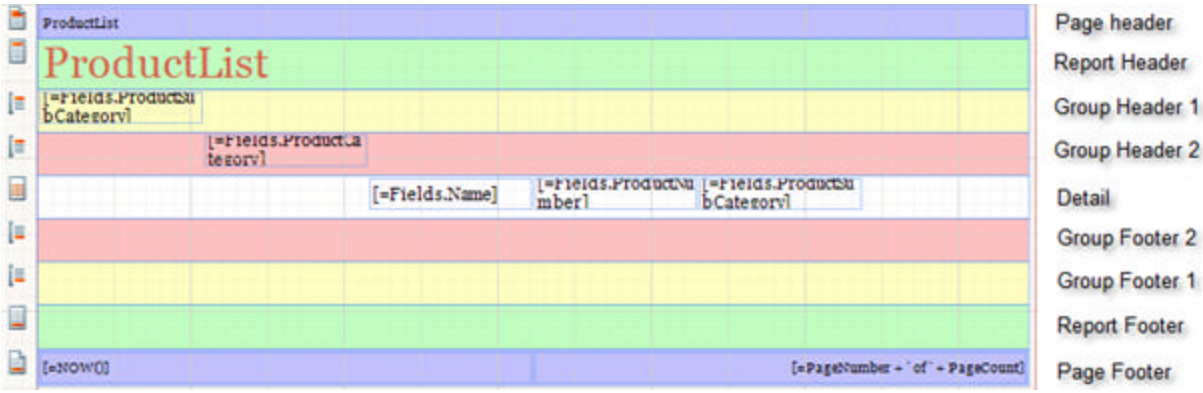


Report Wizard

The **Telerik Report Wizard** helps you quickly create your report structure. You can also convert existing Crystal, ActiveReports or Developer Express XTra Reports to the Telerik Reporting format. The wizard walks you through selecting, arranging and formatting the data for the report. The **Report Wizard** runs automatically when you add a **Telerik Report** item to the project, but can also be run from the Telerik Reporting menu. Each run of the **Report Wizard** overwrites previous definitions of the report.

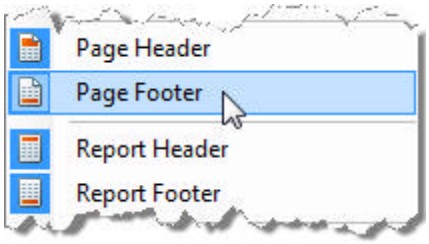
Report Sections

Your report layout will typically include multiple sections and always at least one detail section (the detail section cannot be removed).



Page Header and Footer Sections

Report items placed in the page header section are rendered at the top of each page of report output. Likewise, report items placed in the page footer section render at the bottom of each page of report output. You can add or remove page headers and footers by right-clicking the empty space around the sections and selecting **Page Header** or **Page Footer** from the context menu. You can also select a section with the mouse and click the **Delete** key to remove that section.



Report paging strongly depends on the format it is rendered to. Page header and footer sections are processed by a rendering extension *after* the report data has been processed. At this moment the report data source is not available anymore and you cannot add databound items directly to these sections. For this same reason, you can use the built-in **PageNumber** and **PageCount** objects only in these sections.

Report Header and Footer Sections

Reports can contain headers and footers for the report as a whole.

- Any control you place in the report header section is rendered once at the top of the entire report's output, but after the first page header section (if any).
- Any control you place in the report footer section is rendered once at the bottom of the entire report's output, but before the last page footer section (if any).

Why does the page header print before the report header? A report can be viewed as "document content", consisting of detail, report header and footer sections. Similar to Microsoft Word, document content is centered on the screen with page header and footer sections above and below the content. Page headers and footers are not related to the report itself, but are relative to the margins of the report's media, i.e. to the paper or screen. Usually page headers and footers are placed in the margins and contain page numbers, document dates, etc.

Groups

You can group data in the report so that some change in data triggers a break in the report. For example, if a product listing report has a product category and subcategory, the report can show first the category, then subcategories below it.

Each Group has a **GroupHeaderSection** and **GroupFooterSection** associated with it. The GroupHeaderSection prints at the beginning of each group and typically has a name or description of the group. The GroupFooterSection prints after the detail data for the group and can contain summary data. You can output aggregate functions (e.g. totals, counts, etc). Each group can be filtered and sorted.

In the Group header and footer sections all data fields must be aggregated, even if the data source returns only one row. Typically you should use the FIRST() function for character and date data and the SUM() function for numeric data. When you place a databound report item that uses an aggregate function in the Group Header, it is calculated for the entire group data.

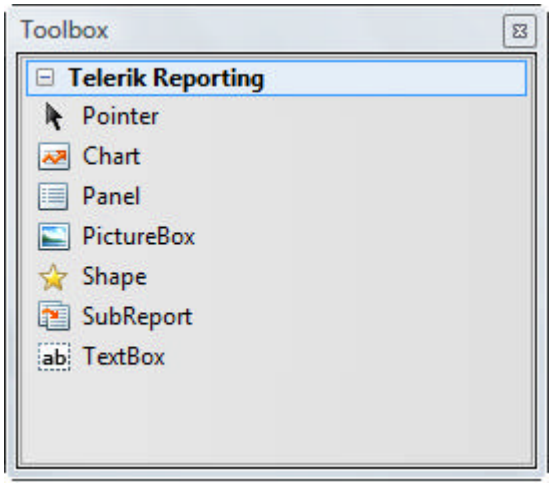
Detail

The Detail section is always present for every report. For reports bound to data, a detail section will output for every database record. This is where you place the report items that make up the main body of the report.

Report Items

Report items are found in the Visual Studio Toolbox window and are dragged onto the report design surface to contain report content.

- **Chart**: The chart lets you display data in a visually compelling way. The chart can bind to live data and display in many popular chart types, e.g. Bar, Pie, Gantt, Line, Area, Bubble, etc.
- **Panel**: The Panel report item is used to group multiple report items together.
- **PictureBox**: The PictureBox allows you to place images in the report using bitmap files (e.g. bmp, gif, jpg, png, ico).
- **Shape**: The Shape item displays visual primitives in the report including ellipse, vertical/horizontal/slanting lines, and triangles.
- **SubReport**: The SubReport report item lets you display one report within another report. This allows you to compose complex reports from disparate report sources.
- **TextBox**: The TextBox displays text in the report, can be styled to a specific visual appearance, can be configured to grow/shrink, formatted for specific data types (e.g. date time) and can be multi-line.



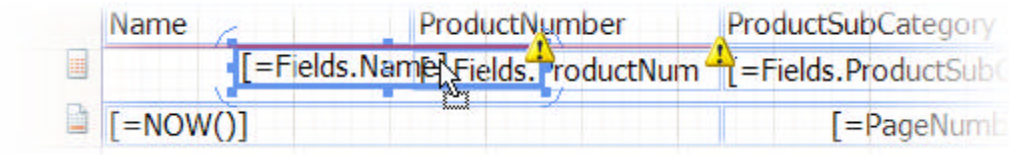
The TextBox in-place editor allows you to quickly enter text directly into the designer where the report item is rather than having to go look for a property in the Property Window. To activate the in-place editor, double-click the **TextBox** report item or select the report item and press **F2**. Once the in-place editor is activated:

- CTRL-Enter creates a new line.
- Enter accepts all changes.
- Esc discards all changes.
- Moving focus away from the **TextBox** also accepts all changes.

Placing, Sizing and Rotating Report Items

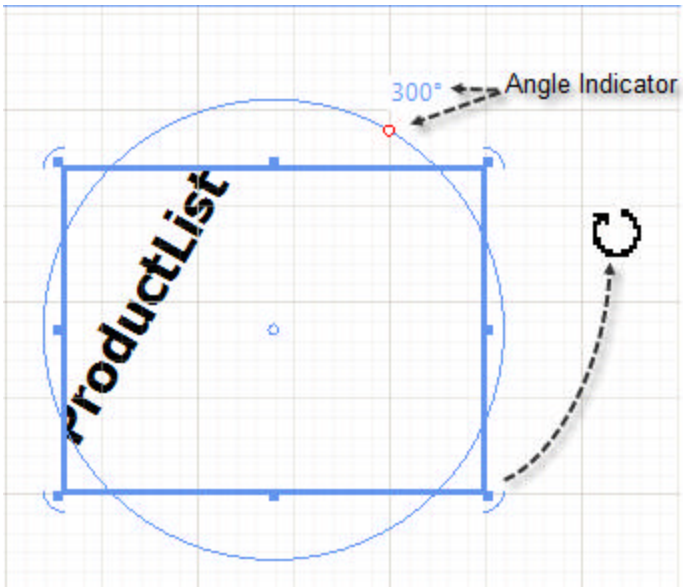
To add report items, you can drag them from the Toolbox window to the report design surface. You can also double-click to add a report item to the currently selected report section. From there you can drag the items in the report section or to other report sections.

Grid lines and snap lines will help orient the report item so that it lines up with other portions of the report. Watch for the visual cues (the yellow hazard icons) that one item is overlapping another.



Grab and drag the resizing handles from any size or corner of the report item:

Notice the small curved lines around each corner. These allow you to drag and rotate a report item. As the item is rotated, a circle appears showing the area that the item is being rotated within. The outside edge of the circle displays a tiny red dot and a figure indicating the number of degrees that the item is being rotated.

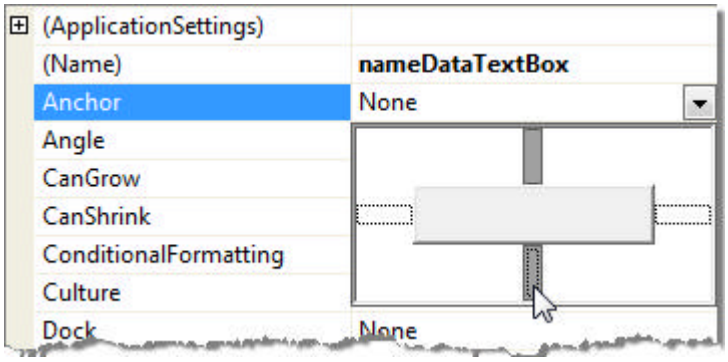


Dynamic Layout

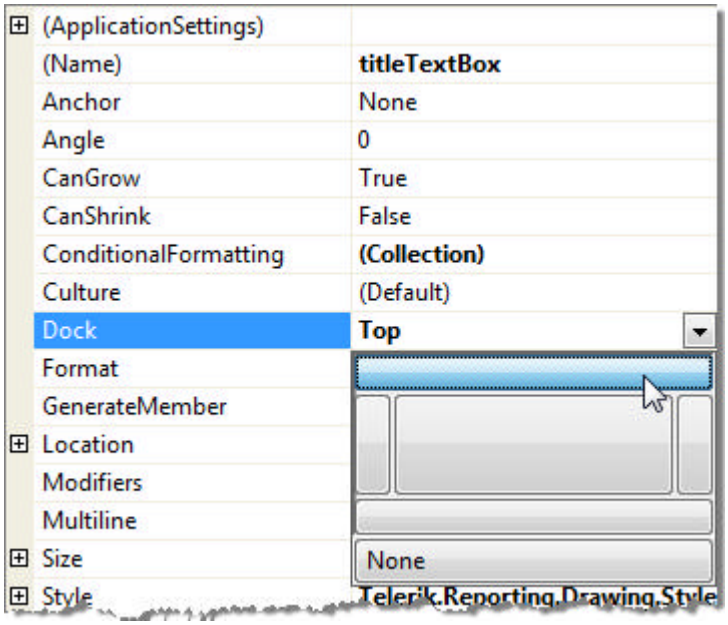
Anchoring and Docking properties have long been available to WinForms applications. Telerik Reports brings this feature to the reporting design environment. Using Anchoring and Docking you can create table-like report layouts – all items inside a section can grow equally regardless of their content thus forming the “columns” of a table. Dynamic layout comes into play when the container can be resized, for example when:

- A child TextBox item with the **CanGrow** property enabled can grow in height to accommodate long strings of text.
- A SubReport item can grow in width and height according to the source report it contains.
- A PictureBox item with auto sizing enabled (PictureBox.SizingMode= AutoSize) can grow to fit a bigger image.

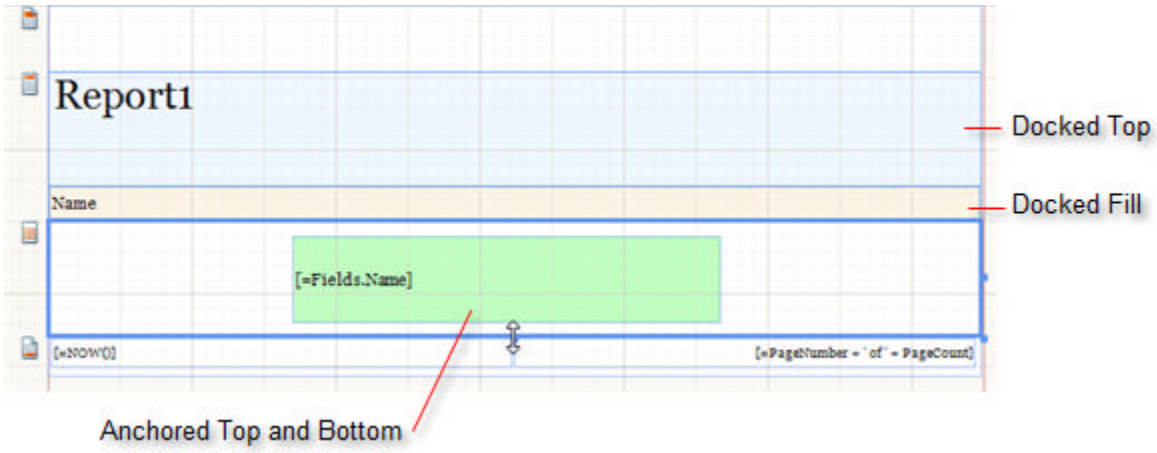
Anchoring is used to dynamically resize report items with their container, i.e. Panel or section. As the container is resized, the report item will maintain a constant distance between its edges and the corresponding edges of the container.



Docking a report item causes its edges to adhere to the edges of its container. When a report item is docked to an edge of its container, it is always positioned flush against that edge, even when the container is resized.



The example below has a page header section with two docked TextBox items, the first docked to the Top of the section, the second docked to Fill the remaining space in the section. In the detail section a TextBox item is anchored with top and bottom edges not quite touching. As the section is resized, the TextBox maintains its relationship to the top and bottom edges.



Summary

We took a tour of the report design environment, the report sections that house content and became familiar with the report items that are placed within the report sections.

5 Connecting to Data

Objectives

- Learn what types of data can be bound.
- Bind to data manually.
- Bind to data programmatically.

Data Source Types

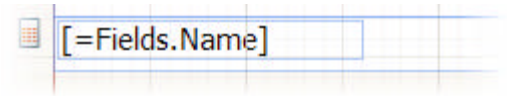
The lion's share of business reporting applications will consume data from a database. The report **DataSource** property can be assigned any component that implements **IEnumerable**, **IListSource**, **ITypedList** or **IDbDataAdapter**:

- **IEnumerable**: e.g. **System.Array**, **IList**, **ICollection**, and **BindingSource**
- **IListSource**: e.g. **DataTable** and **DataSet**
- **ITypedList**: e.g. **DataView** and **DataViewManager**
- **IDbDataAdapter**: e.g. **SqlDataAdapter** and **OleDbDataAdapter**

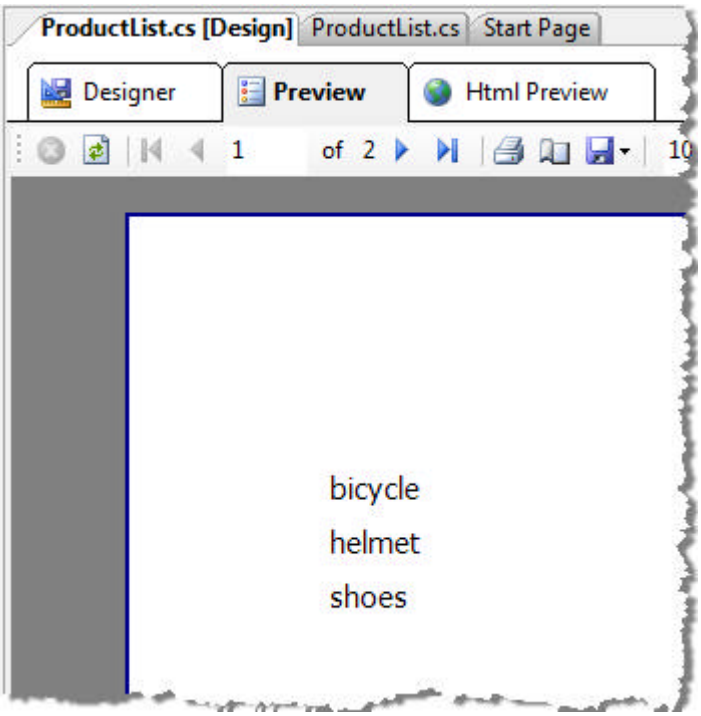
> Telerik Reporting does not have its own data layer but depends on the existing .NET objects (DataSet, Data Table, DataView, ADO.NET, lists etc.) For example, if you are connecting to a stored procedure you will end up returning an object that can be used as a data source, e.g. a DataSet, DataView, DataReader, etc. For more specifics on connecting to stored procedures please see Microsoft knowledge base article **Call a Parameterized Stored Procedure by Using ADO.NET... (http://support.microsoft.com/kb/310070)**

You can assign these data source types through the Report Wizard, manually at design time or programmatically at run time. At the simplest level you could create an array or generic list of a simple object and bind it to the report.

For example, let's say we have a report definition with only a detail section. The detail section contains only a single Item Binding Expression. We will cover Item Binding Expressions shortly, but for now, see the expression in the screenshot below "=Fields.Name".



When the report is bound, the data for the "Name" property of the data source will be output.



The code to get this done can be as simple as the example below where a generic list is created, populated and assigned to the report **DataSource** property. The generic list contains a series of Product objects. "Product" is the most minimal class possible and contains only a single "Name" property.

[C#] Populating and Assigning a DataSource

```
using System.Collections.Generic;
public partial class ProductList : Report
{
   public ProductList()
   {
       InitializeComponent();
       // Create a generic list of Product and
       // add sample products
       List<Product> products = new List<Product>();
       products.Add(new Product("bicycle"));
       products.Add(new Product("helmet"));
       products.Add(new Product("shoes"));
       // Assign the generic list as the report DataSource
       this.DataSource = products;
   }
}
// A very simple class with a single Name property
public class Product
{
   private string _name;
   public Product(string name)
   {
       _name = name;
   }
   public string Name
   {
       get { return _name; }
```

```
        set { _name = value; }
    }
}
```

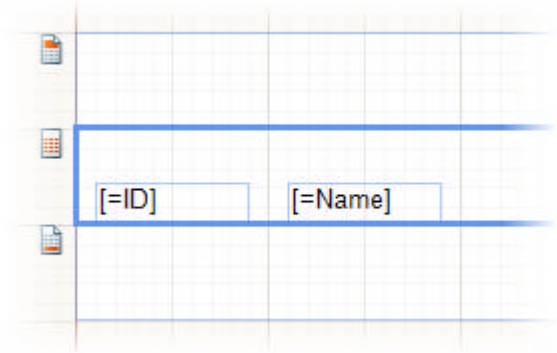
[VB] Populating and Assigning a DataSource

```
Imports System.Collections.Generic
Public Partial Class ProductList
 Inherits Report
 Public Sub New()
  InitializeComponent()
  ' Create a generic list of Product and
  ' add sample products
  Dim products As New List(Of Product)()
  products.Add(New Product("bicycle"))
  products.Add(New Product("helmet"))
  products.Add(New Product("shoes"))
  ' Assign the generic list as the report DataSource
  Me.DataSource = products
 End Sub
End Class
' A very simple class with a single Name property
Public Class Product
 Private _name As String
 Public Sub New(ByVal name As String)
  _name = name
 End Sub
 Public Property Name() As String
  Get
   Return _name
  End Get
  Set
   _name = value
  End Set
 End Property
End Class
```

Lab: Binding to Array Data

Later you will learn how to use more conventional database data as the data source. This walk-through will show you how to hook up a simple array of objects as a data source for your report.

1. Create a class library and a report class similar to the example in "Creating Simple Output" (be sure to cancel the Report Wizard).
2. Place two TextBox report items side-by-side in the Detail section of the designer.
3. Double-click the first TextBox report item and enter "=ID".
4. Double-click the second TextBox report item and enter "=Name".



5. Right-click the designer and select "View Code" from the context menu.
6. Add a "Coffee" class:

[C#] Simple Class With Two Properties

```
public class Coffee
{
   private string _name;
   private int _id;
   public Coffee(string name, int id)
   {
       _name = name;
       _id = id;
   }
   public string Name
   {
       get { return _name; }
       set { _name = value; }
   }
   public int ID
   {
       get { return _id; }
       set { _id = value; }
   }
}
```

[VB] Simple Class With Two Properties

```
Public Class Coffee
 Private _name As String
 Private _id As Integer
 Public Sub New(ByVal name As String, ByVal id As Integer)
  _name = name
  _id = id
 End Sub
 Public Property Name() As String
  Get
   Return _name
  End Get
  Set
   _name = value
  End Set
 End Property
 Public Property ID() As Integer
  Get
   Return _id
```

```
  End Get
  Set
   _id = value
  End Set
 End Property
End Class
```

7. Create an array of "Coffee" and assign the array to the report data source.

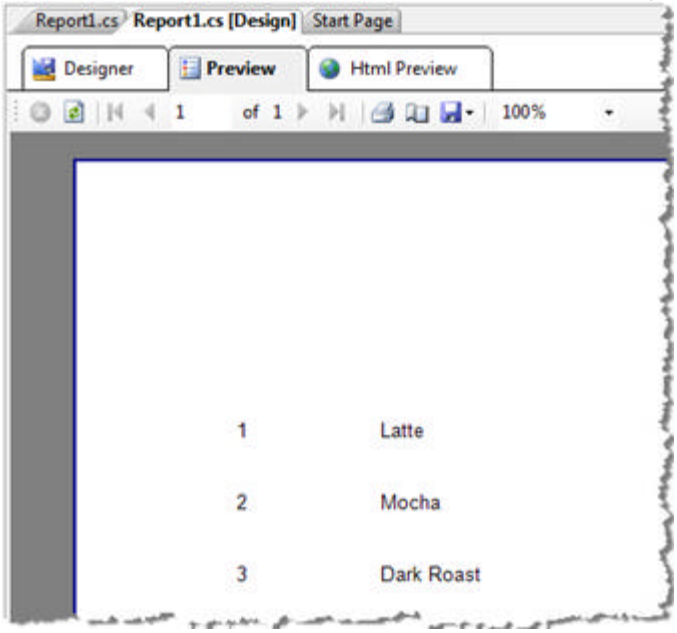
   [C#] Create and Populate Coffee Array

```
public Report1()
{
   InitializeComponent();
   // create and populate an array of Coffee objects
   Coffee[] coffees = new Coffee[]
   {
       new Coffee("Latte", 1),
       new Coffee("Mocha", 2),
       new Coffee("Dark Roast", 3)
   };
   // assign the array to the DataSource
   this.DataSource = coffees;
}
```

   [VB] Create and Populate Coffee Array

```
Public Sub New()
 InitializeComponent()
 ' create and populate an array of Coffee objects
 Dim coffees As Coffee() = New Coffee() {New Coffee("Latte", 1), New Coffee("Mocha", 2),
New Coffee("Dark Roast", 3)}
 ' assign the array to the DataSource
 Me.DataSource = coffees
End Sub
```

8. Navigate back to the design view by clicking the report design tab in Visual Studio.
9. Click the **Preview** button tab. You should see the array ID and Name properties output to the report.

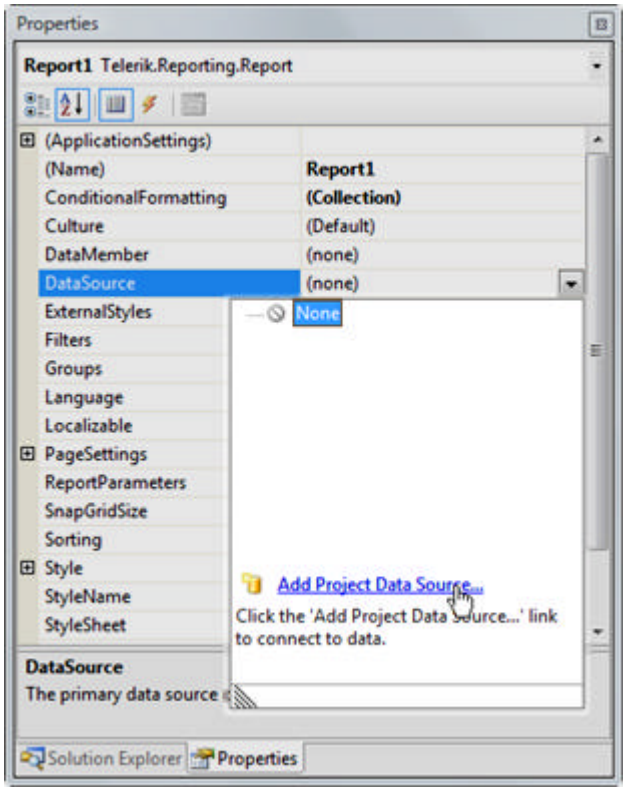


Lab: Binding to Data Manually

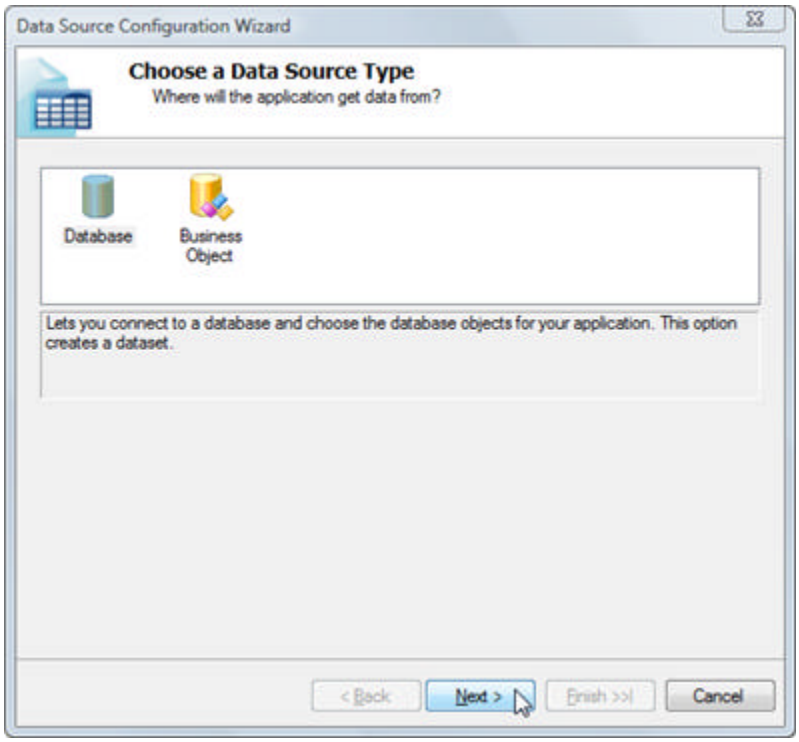
A typical business scenario consumes data from a standard database. For example, MS SQL, Oracle, My SQL, Access or just about any data you can hook up to through OleDb can be data sources for reports.

You can assign a data source using the Report Wizard as we did in the "Creating a Simple database Report" lab or you can use the **DataSource** property to invoke the **Data Source Configuration Wizard**. The following walk-through demonstrates hooking up AdventureWorks address data and using the Data Explorer to construct a report from the data.

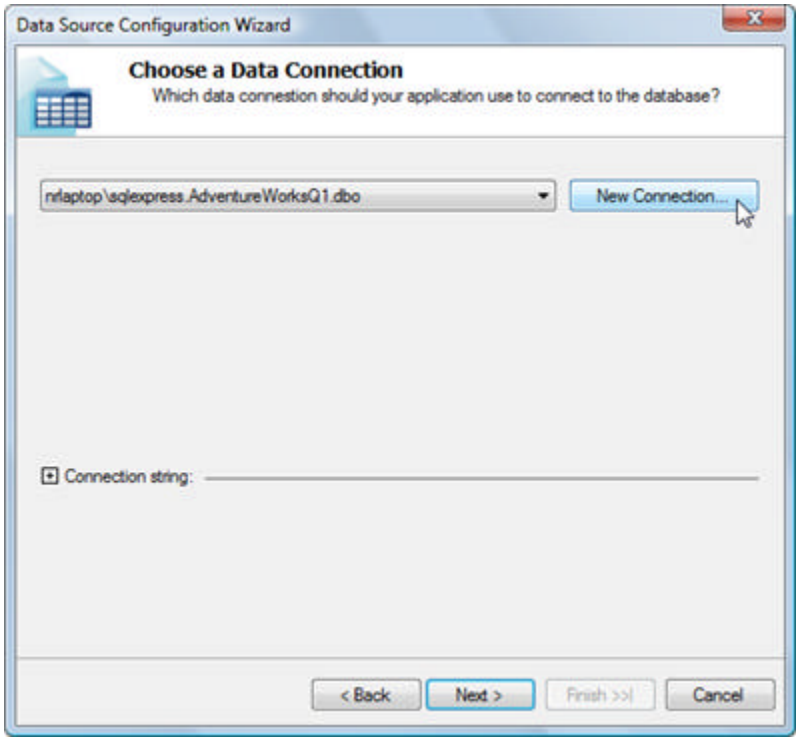
1. Create a class library and a report class similar to the example in "Creating Simple Output" (be sure to cancel the Report Wizard).
2. Click on the report in the design surface (in the area outside of the report sections).
3. In the Properties window, locate the report **DataSource** property, click the arrow to drop down the property editor and select **Add Project Data Source...**



4. The Data Source Configuration Wizard starts at the **Choose a Data Source Type** page. Here you can choose between a standard database (e.g. MS SQL, Access, Oracle, etc.) or a Business Object that acts as a middle tier for data. In this example, click the **Database** icon, then click the **Next** button.

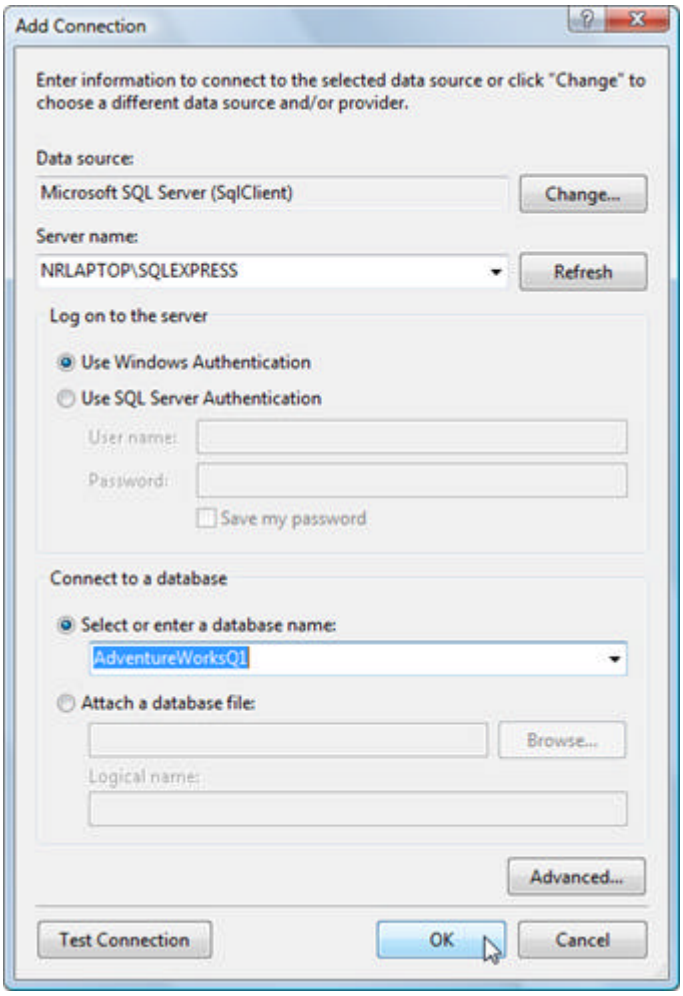


5. In the **Choose a Data Connection** page of the wizard, you can select a previous connection from the drop down list or create a new connection. Click the **New Connection** button.

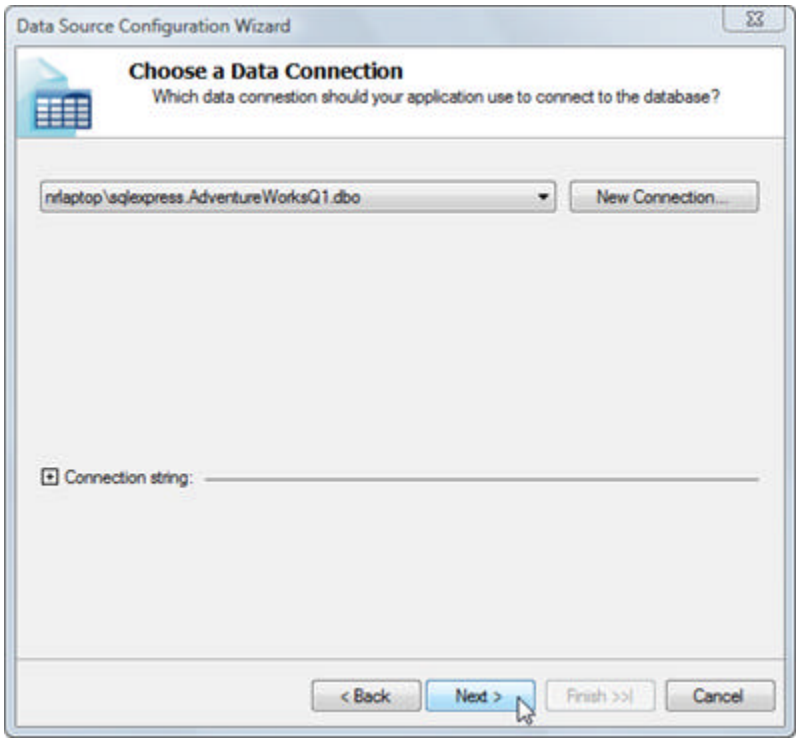


6. In the **Add Connection** dialog, verify that the **Data Source** is "Microsoft SQL Server (SqlClient)". If not, click the **Change** button to select a different data source provider. To the **Server Name** entry add "<your machine name>\SQLEXPRESS". In the **Connect to a database** box, make sure the radio button labled "Select or enter a database name:" is selected, drop down the list and select the "AdventureWorks" database. You

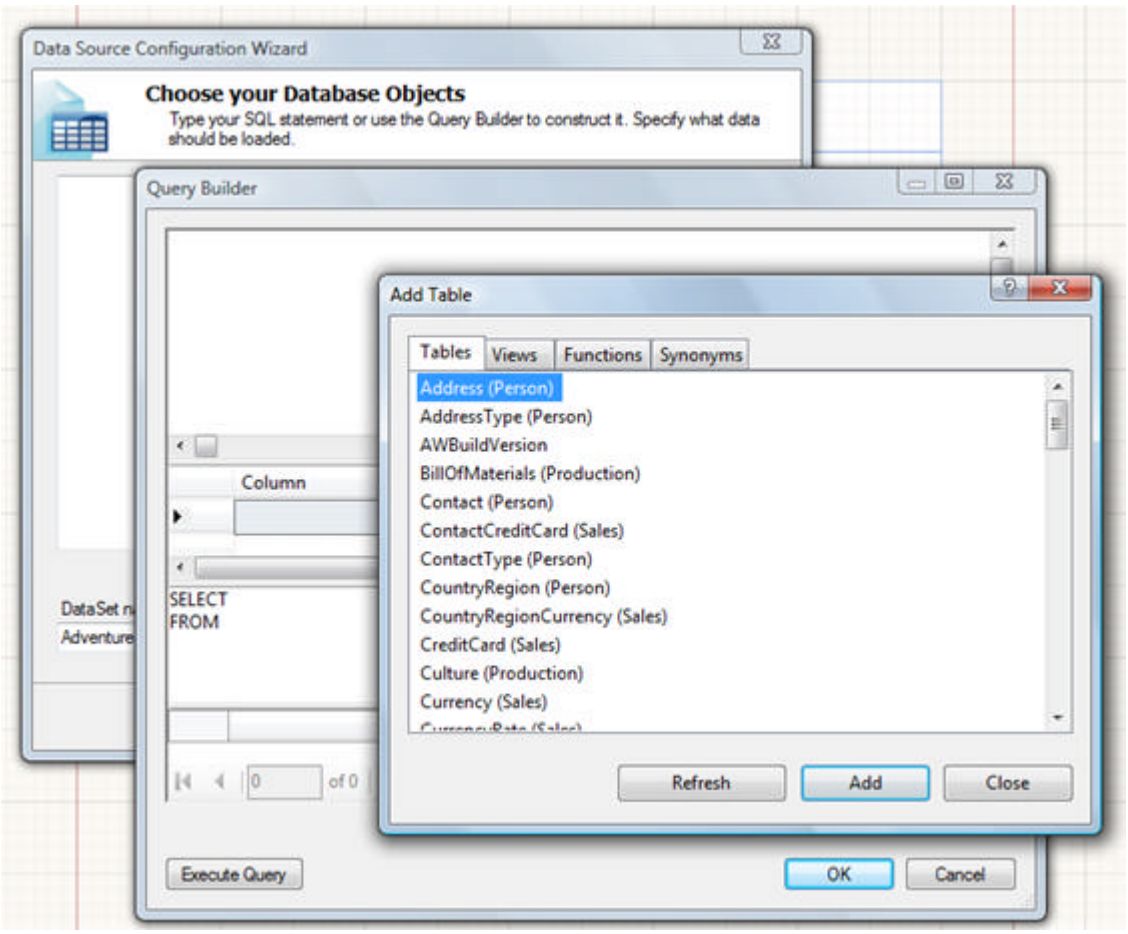
can click the **Test Connection** button to verify that everything works. Click **OK** to close the Add Connection dialog.



7. Closing the **Add Connection** dialog brings you back to the **Choose a Data Connection** page of the wizard. Make sure that the connection you created is selected from the drop down list. Click the **Next** button to continue.



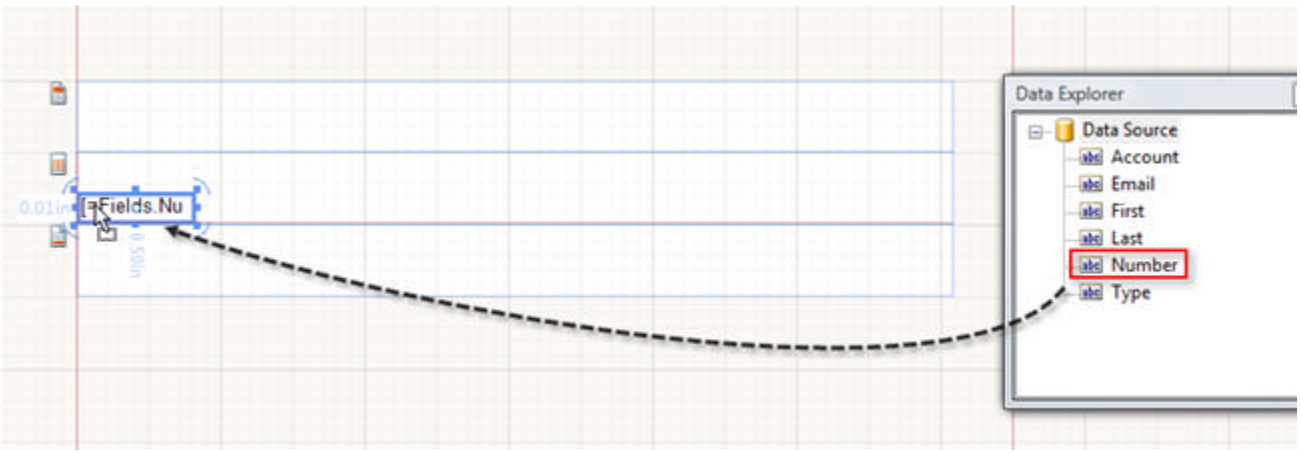
8. In the **Choose Your Database Objects** page of the wizard, you can enter SQL directly or click the **Query Builder.** The **Query Builder** dialog automatically displays the **Add Table** dialog. From there you can select one or more tables, views, functions or synonyms to be included in the query. Query Builder also has an **Execute Query** button so you can "audition" the data.



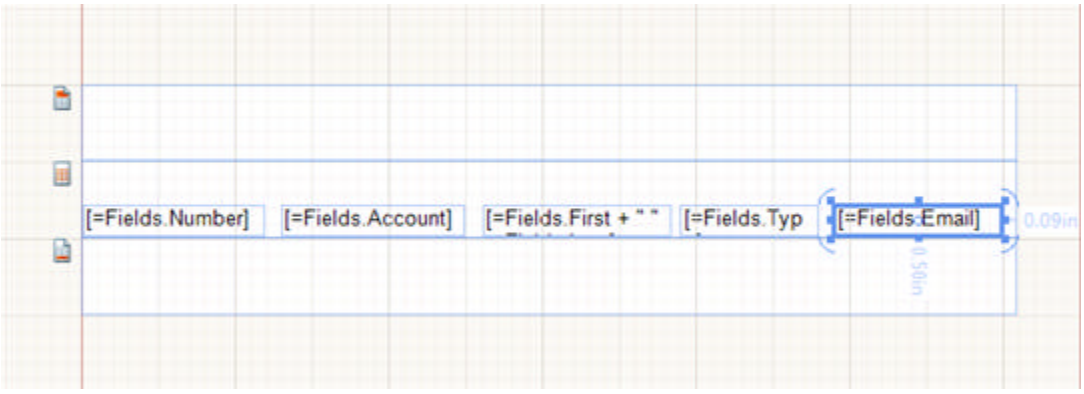
9. In this case, add the following SQL to the **Choose Your Database Objects** page of the wizard:
   [SQL] Adding the Vendor Contacts Query

```
SELECT
 Purchasing.Vendor.AccountNumber AS [NUMBER],
 Purchasing.Vendor.NAME AS [Account],
 Person.Contact.FirstName AS FIRST,
 Person.Contact.LastName AS LAST,
 Person.Contact.EmailAddress AS Email,
 Person.ContactType.NAME AS [TYPE]
FROM Purchasing.VendorContact
INNER JOIN
Purchasing.Vendor ON Purchasing.VendorContact.VendorID = Purchasing.Vendor.VendorID
INNER JOIN
Person.Contact ON Purchasing.VendorContact.ContactID = Person.Contact.ContactID
INNER JOIN
Person.ContactType ON Purchasing.VendorContact.ContactTypeID =
Person.ContactType.ContactTypeID
```

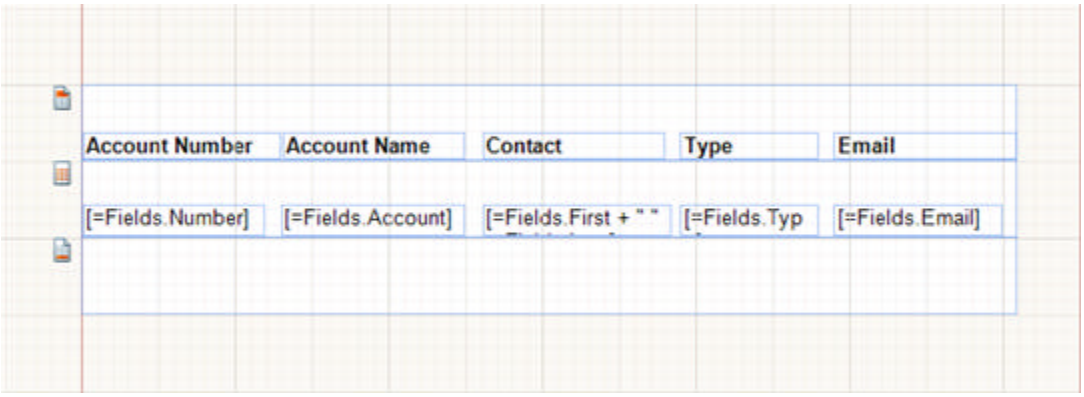
10. Click the **Finish** button to complete and close the wizard.
11. Open the Telerik Reporting menu and select the **Data Explorer**. **Note**: if you don't see the Telerik Reporting menu, click on the report design surface - the menu will appear.
12. Drag fields "Number" and "Account" to the Detail section of the report.



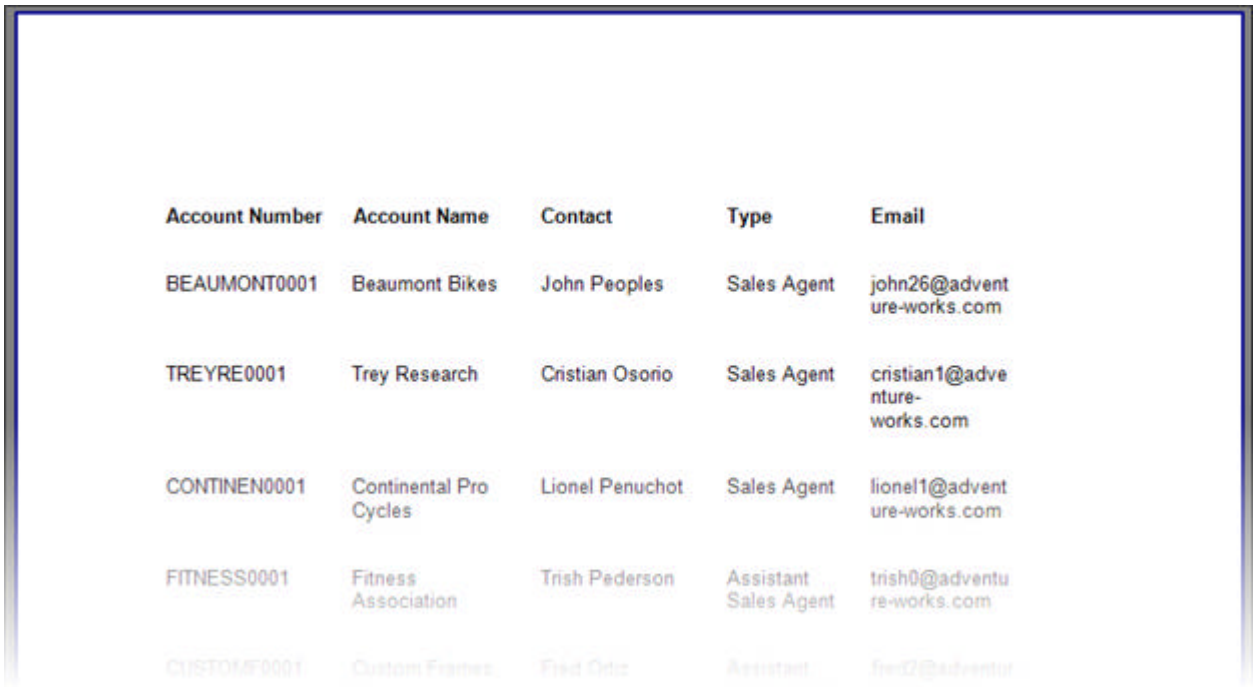
13. Drag a **TextBox** report item from the ToolBox window to the Detail section of the report.
14. Double-click the TextBox and enter **'=Fields.First + " " + Fields.Last'.** This will output the concatenated first and last names of the contact name.
15. Drag the other fields "Type" and "Email" to the Detail section of the report.



16. Now drag TextBox report items from the ToolBox window to the Page Header section of the report. Drag one TextBox per field and name the columns "Account Number", "Account Name", "Contact", "Type" and "Email".



17. Click the **Preview** tab to view the report. Notice that the Page Header text outputs at the top of each page, with multiple detail lines on each page. Also notice that the contact name has the concatenated first and last names.



Account Number	Account Name	Contact	Type	Email
BEAUMONT0001	Beaumont Bikes	John Peoples	Sales Agent	john26@adventure-works.com
TREYRE0001	Trey Research	Cristian Osorio	Sales Agent	cristian1@adventure-works.com
CONTINEN0001	Continental Pro Cycles	Lionel Penuchot	Sales Agent	lionel1@adventure-works.com
FITNESS0001	Fitness Association	Trish Pederson	Assistant Sales Agent	trish0@adventure-works.com

Binding to Data Programmatically

You can also bind to data from a data source created and populated at runtime. The example below uses objects from the **System.Data.SqlClient** namespace, but the same general pattern will also work with objects from the **System.Data.OleDb** namespace. The example below creates a Sql connection, command and data adapter, runs the command and assigns the adapter to the report data source.

[C#] Generating and Assigning MS SQL Data at Runtime

```
using System.Data.SqlClient;
//...
public Report1()
{
   InitializeComponent();
   // create connection, command, adapter
   SqlConnection sqlConnection1 = new SqlConnection();
   SqlCommand sqlSelectCommand1 = new SqlCommand();
   SqlDataAdapter sqlDataAdapter1 = new SqlDataAdapter();
   // define a connection string. Note that you may have to change the
   // "Initial Catalog" portion of the string depending on your current installation.
   sqlConnection1.ConnectionString =
       "Data Source=(local)\\SQLEXPRESS;Initial Catalog=AdventureWorksQ1;Integrated
Security=True";
   // define the command object properties, assign to the
   sqlSelectCommand1.CommandText = "SELECT * FROM Production.Product";
   sqlSelectCommand1.Connection = sqlConnection1;
   sqlDataAdapter1.SelectCommand = sqlSelectCommand1;
   this.DataSource = sqlDataAdapter1;
}
```

[VB] Generating and Assigning MS SQL Data at Runtime

```
Imports System.Data.SqlClient;
'...
```

```
Public Sub New()
 InitializeComponent()
 ' create connection, command, adapter
 Dim sqlConnection1 As New SqlConnection()
 Dim sqlSelectCommand1 As New SqlCommand()
 Dim sqlDataAdapter1 As New SqlDataAdapter()
 ' define a connection string. Note that you may have to change the
 ' "Initial Catalog" portion of the string depending on your current installation.
 sqlConnection1.ConnectionString = "Data Source=(local)\SQLEXPRESS;Initial
Catalog=AdventureWorksQ1;Integrated Security=True"
 ' define the command object properties, assign to the
 sqlSelectCommand1.CommandText = "SELECT * FROM Production.Product"
 sqlSelectCommand1.Connection = sqlConnection1
 sqlDataAdapter1.SelectCommand = sqlSelectCommand1
 Me.DataSource = sqlDataAdapter1
End Sub
```

Summary

This section defined the underlying types of data that can be bound. You learned how to bind very simple array type data. You also learned how to bind database data manually and programmatically.

6 Item Binding Expressions

Objectives

- Learn the basics of item binding expressions: the types of expressions available and how to assign them to report item values.
- Find out how user functions can extend the Telerik Reporting engine.

Using Item Binding Expressions

Item binding expressions specify what to display in a TextBox or a PictureBox report item. You can assign an expression by entering directly into a TextBox report item in the designer, by modifying the **Value** property or by invoking the **Edit Expression** dialog. To enter an expression directly into a TextBox, type the "=" sign followed by an expression. If a **Value** property is assigned a string starting with the equal (=) sign it will be evaluated as an expression. Some expression examples:

[=Fields.ProductName]

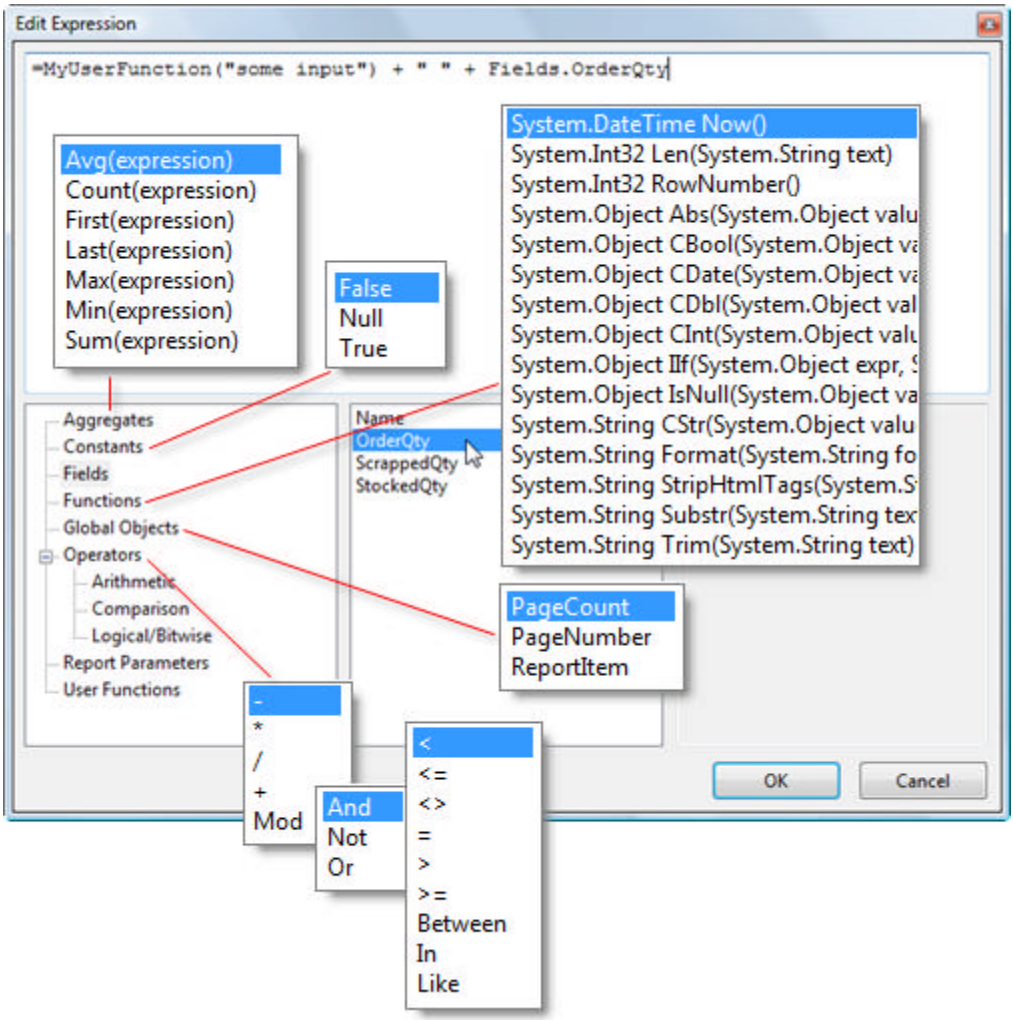
[=10 * 2]

[=Avg(Fields.Quantity)]

[= PageNumber + " of " + PageCount]

To use the Edit Expression dialog, click the ellipses found in the **Value** property of a report item. You can also right-click the report item and select **Expression** from the context menu.

You can type expressions directly into the Edit Expression dialog entry area at the top of the dialog, or use the dialog UI to build the expression. In the lower left hand corner of the dialog is a set of expression types, e.g. Aggregates, Constants, etc. As you click on each of these, the possible expressions appear in a list to the right. For instance, if you click on the **Fields** expression type, a list of fields display. In the screenshot below the fields display as "Name", "OrderQty"... Double-click a field to insert it.



> ⚠ Be aware that not all expressions are available in all sections of the report. For example, PageCount and PageNumber are only available in the page header and footer. These issues will be discussed in more detail during the "Report Life Cycle" discussion that follows later in this tutorial.

User Functions

If the rich set of built-in expressions are not getting the job done, you can create your own user function for consumption directly in your expressions. User functions are defined per report, can be used in any expression (item value, filters, grouping, sorting, conditional formatting) and can be used at design time or run time.

User functions are **public static** (**Public Shared** in VB.NET) methods of the report class that always return a value. As long as the function is public, static and returns a value, you have complete flexibility in your implementation.

The example below formats an address based on city/state/zip passed in as parameters.

[C#] Defining a User Function

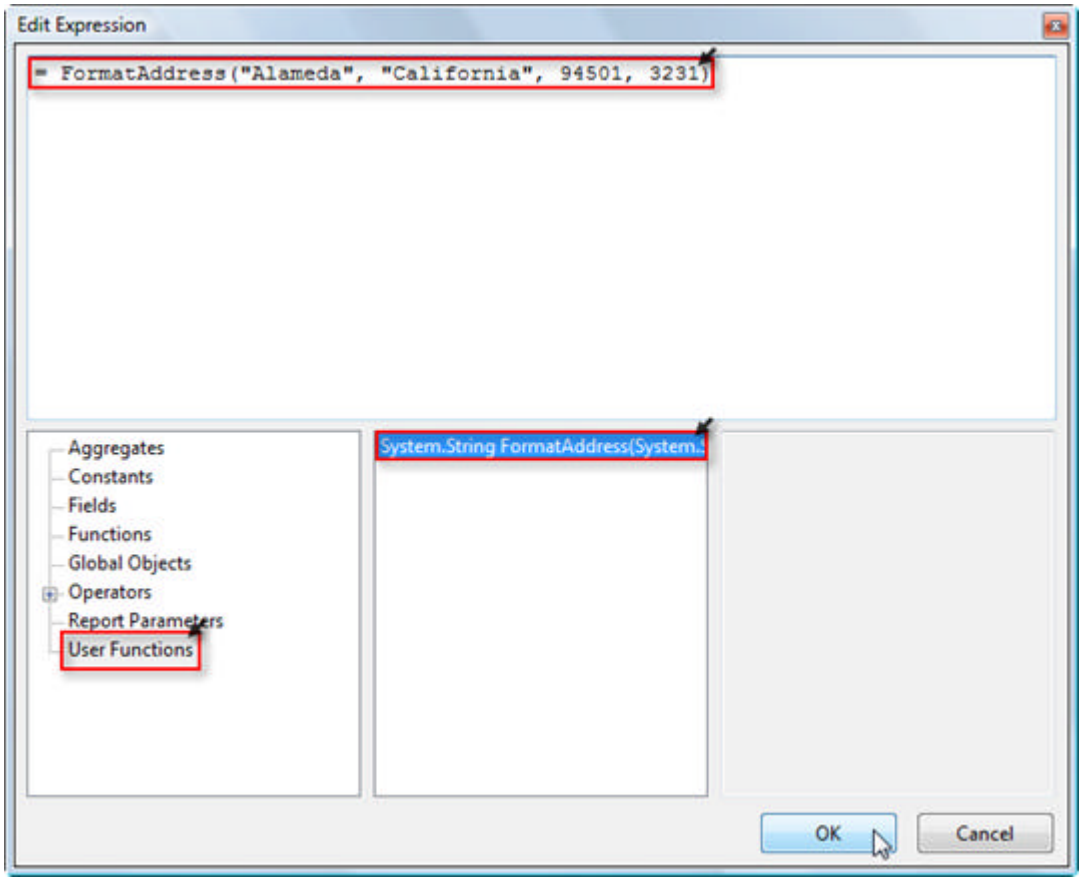
```
public partial class Report1 : Report
{
 //..
 public static string FormatAddress(string city, string state, int zip, int zipPostfix)
 {
```

```
    return String.Format("{0},{1},{2}-{3}", city, state, zip, zipPostfix);
 }
}
```

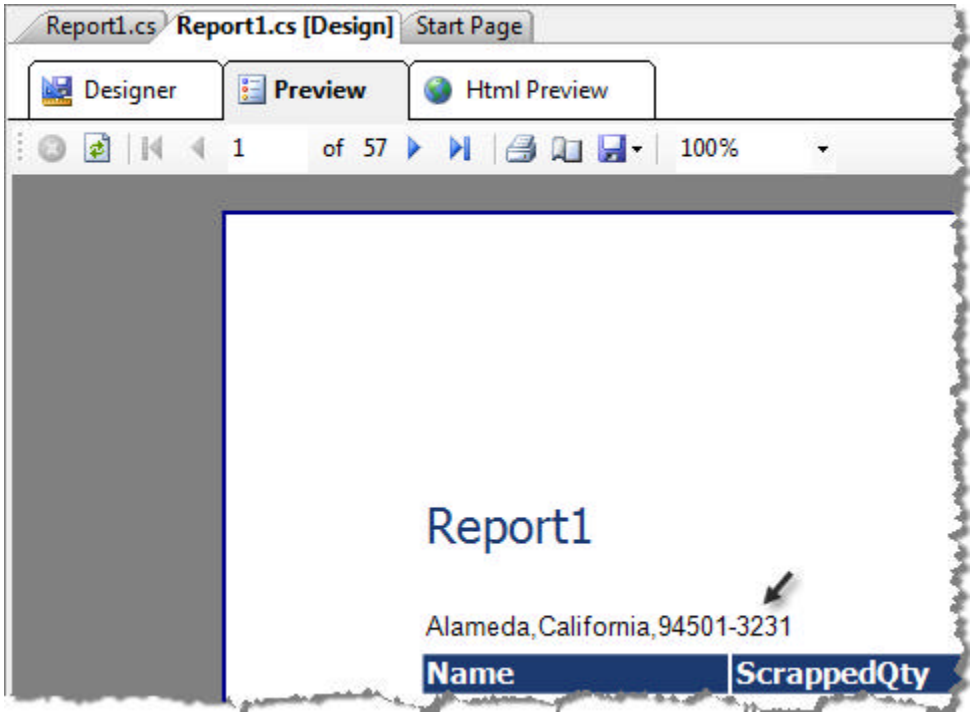
[VB] Defining a User Function

```
Public Partial Class Report1
 Inherits Report
 '..
 Public Shared Function FormatAddress(ByVal city As String, ByVal state As String, ByVal
zip As Integer, ByVal zipPostfix As Integer) As String
  Return [String].Format("{0},{1},{2}-{3}", city, state, zip, zipPostfix)
 End Function
End Class
```

User functions appear at design time in the Edit Expression dialog once the function is defined and the project is compiled.



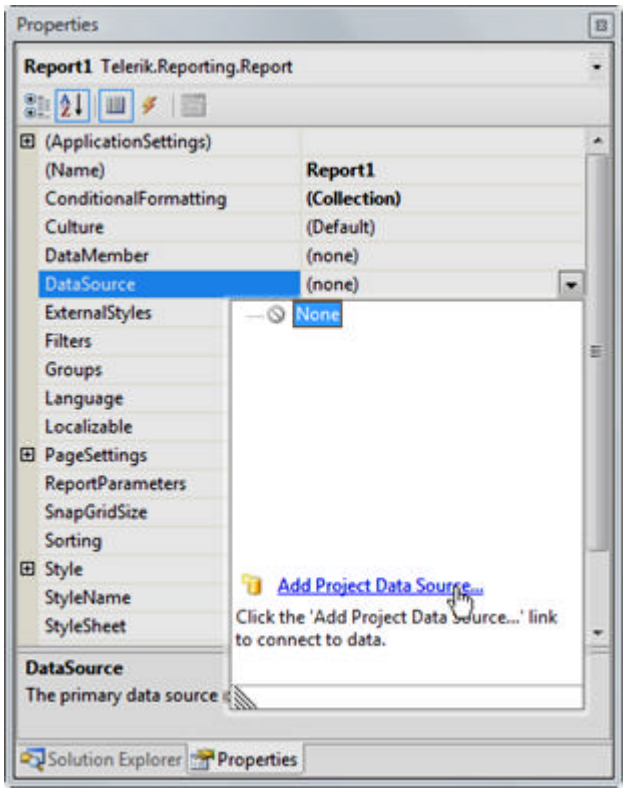
The results of the user function in an expression can be viewed immediately in the Preview tab.



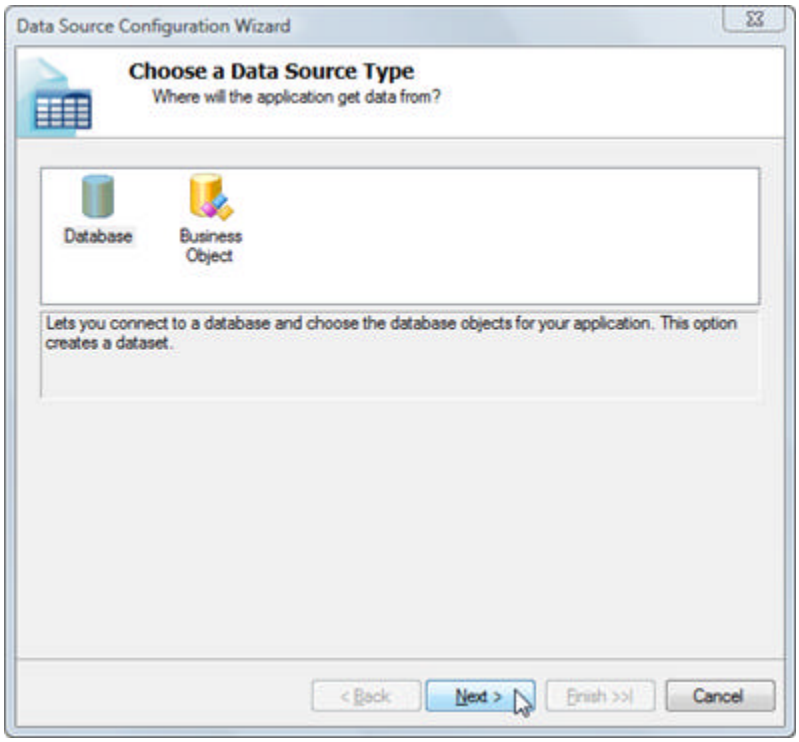
Lab: Using item Binding Expressions

This walk through demonstrates how expressions use bound data, calculations, aggregates and user functions.

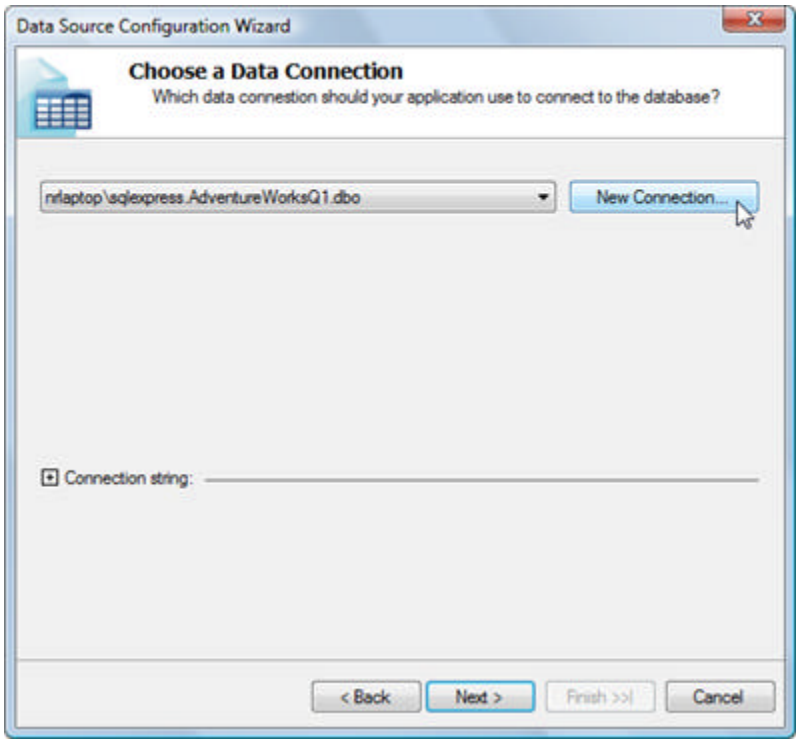
1. Create a class library and a report class similar to the example in "Creating Simple Output" (be sure to cancel the Report Wizard).
2. Select the report in the designer.
3. In the Properties window, locate the report **DataSource** property, click the arrow to drop down the property editor and select **Add Project Data Source...**



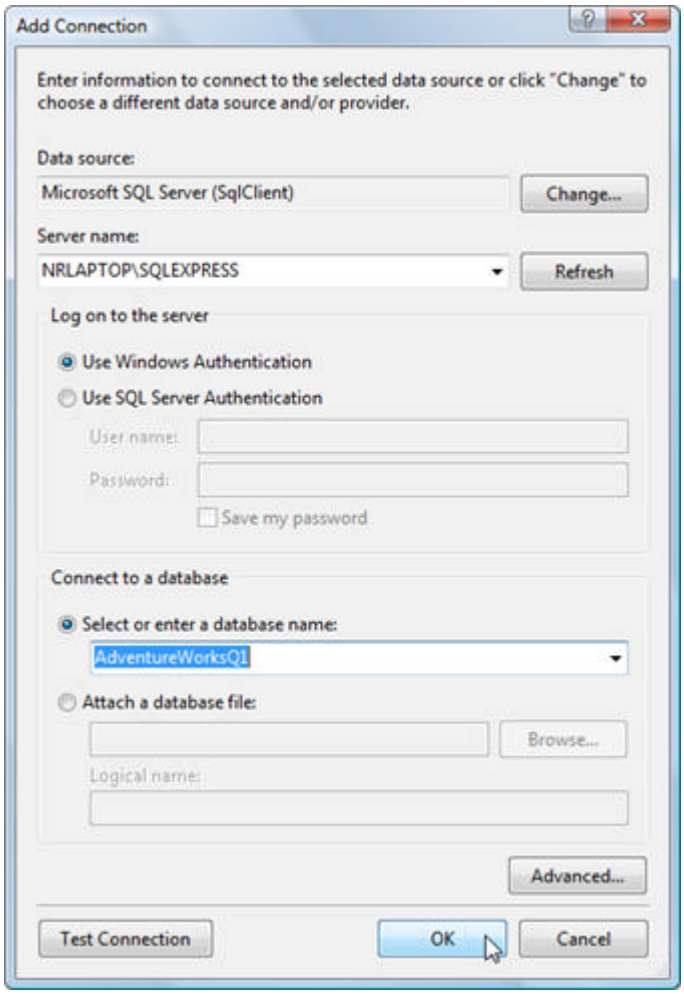
4. The Data Source Configuration Wizard starts at the **Choose a Data Source Type** page. Here you can choose between a standard database (e.g. MS SQL, Access, Oracle, etc.) or a Business Object that acts as a middle tier for data. In this example, click the **Database** icon, then click the **Next** button.



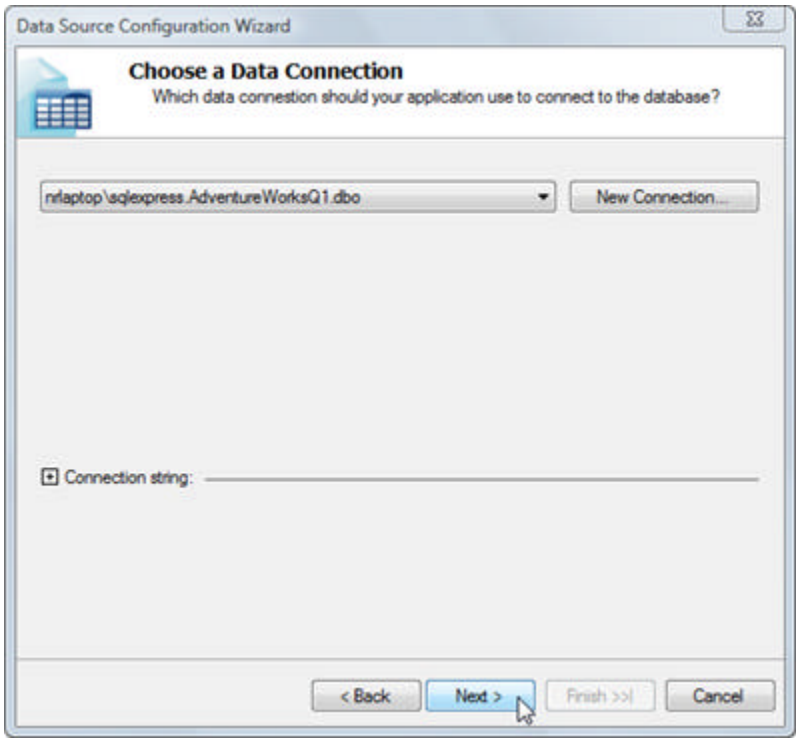
5. In the **Choose a Data Connection** page of the wizard, you can select a previous connection from the drop down list or create a new connection. Click the **New Connection** button.



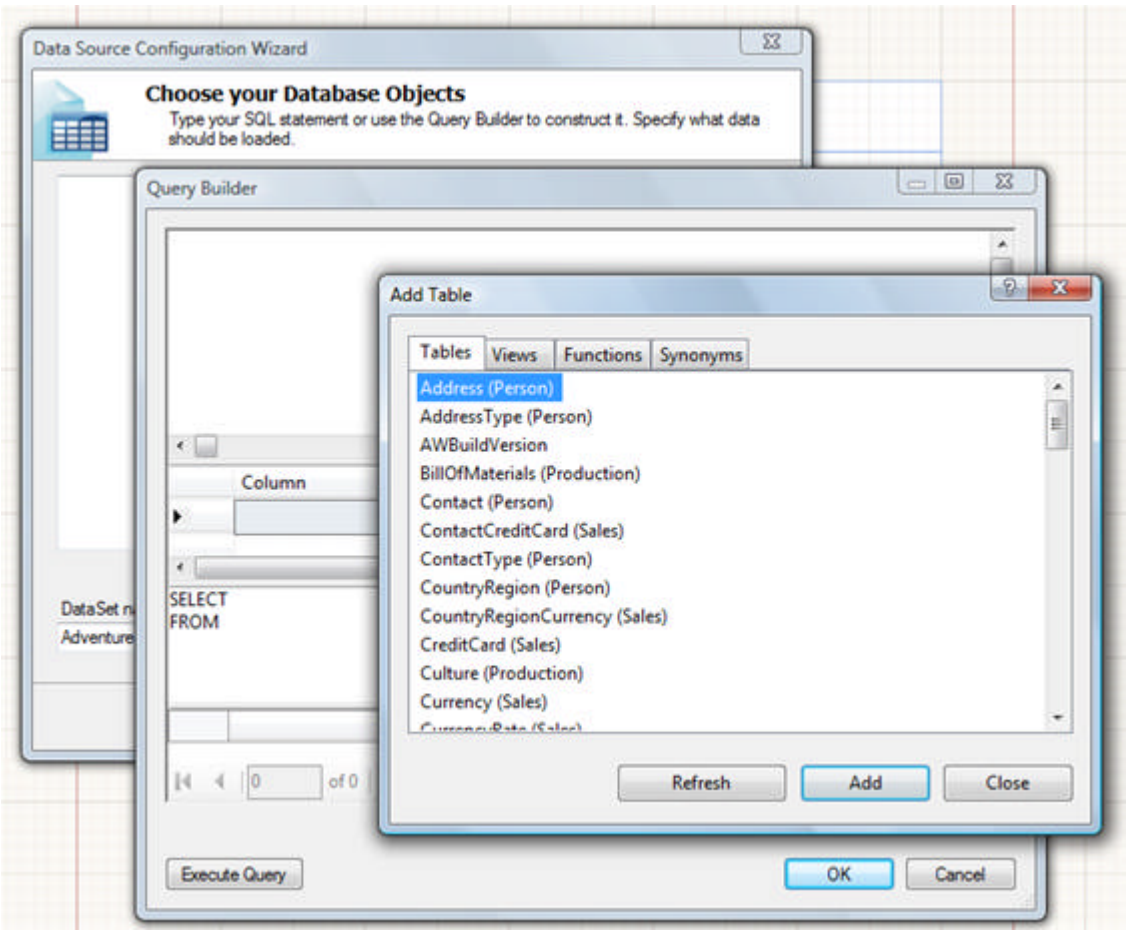
6. In the **Add Connection** dialog, verify that the **Data Source** is "Microsoft SQL Server (SqlClient)". If not, click the **Change** button to select a different data source provider. To the **Server Name** entry add "<your machine name>\SQLEXPRESS". In the **Connect to a database** box, make sure the radio button labled "Select or enter a database name:" is selected, drop down the list and select the "AdventureWorks" database. You can click the **Test Connection** button to verify that everything works. Click **OK** to close the Add Connection dialog.



7. Closing the **Add Connection** dialog brings you back to the **Choose a Data Connection** page of the wizard. Make sure that the connection you created is selected from the drop down list. Click the **Next** button to continue.



8. In the **Choose Your Database Objects** page of the wizard, you can enter SQL directly or click the **Query Builder.** The **Query Builder** dialog automatically displays the **Add Table** dialog. From there you can select one or more tables, views, functions or synonyms to be included in the query. Query Builder also has an **Execute Query** button so you can "audition" the data.



9. Add the following SQL to the **Choose Your Database Objects** page of the wizard:

   [SQL] List Vendor Contacts

```
SELECT top 20
Production.Product.NAME,
Production.WorkOrder.StockedQty,
Production.WorkOrder.ScrappedQty,
Production.WorkOrder.OrderQty
FROM
 Production.Product
INNER JOIN
 Production.WorkOrder ON Production.Product.ProductID = Production.WorkOrder.ProductID
WHERE Production.WorkOrder.ScrappedQty > 0
```

10. Click the **Finish** button to complete and close the wizard.

11. Right-click the report designer and select **View Code** from the context menu. Add the following user function to the report class:

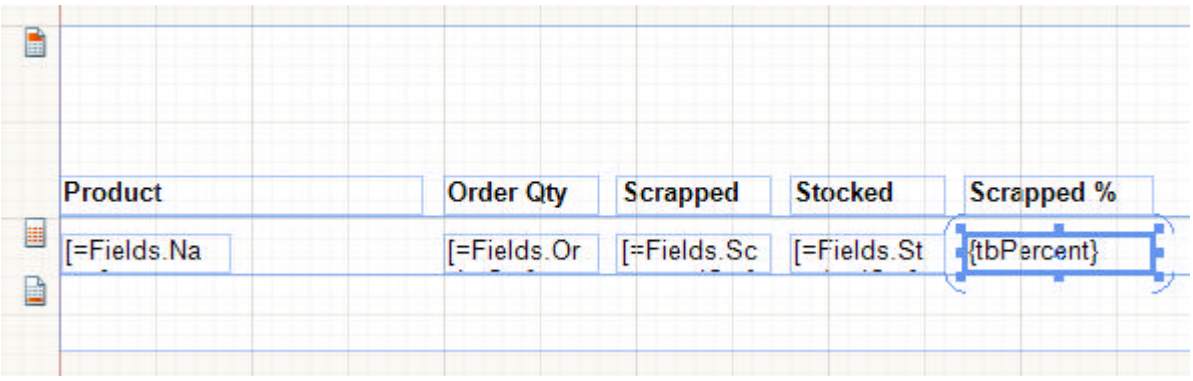
    [C#] Defining the User Function

```
public static double ScrappedPercent(int totalQty, short scrappedQty)
{
   return Convert.ToDouble(scrappedQty) / Convert.ToDouble(totalQty) ;
}
```

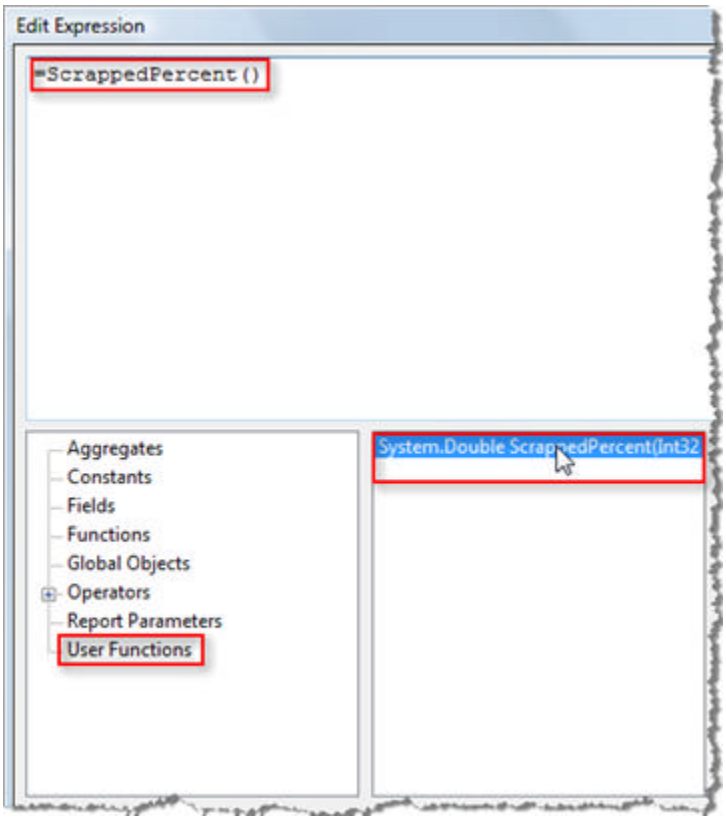
[VB] Defining the User Function

```
Public Shared Function ScrappedPercent(ByVal totalQty As Integer, ByVal scrappedQty As
Short) As Double
 Return Convert.ToDouble(scrappedQty) / Convert.ToDouble(totalQty)
End Function
```

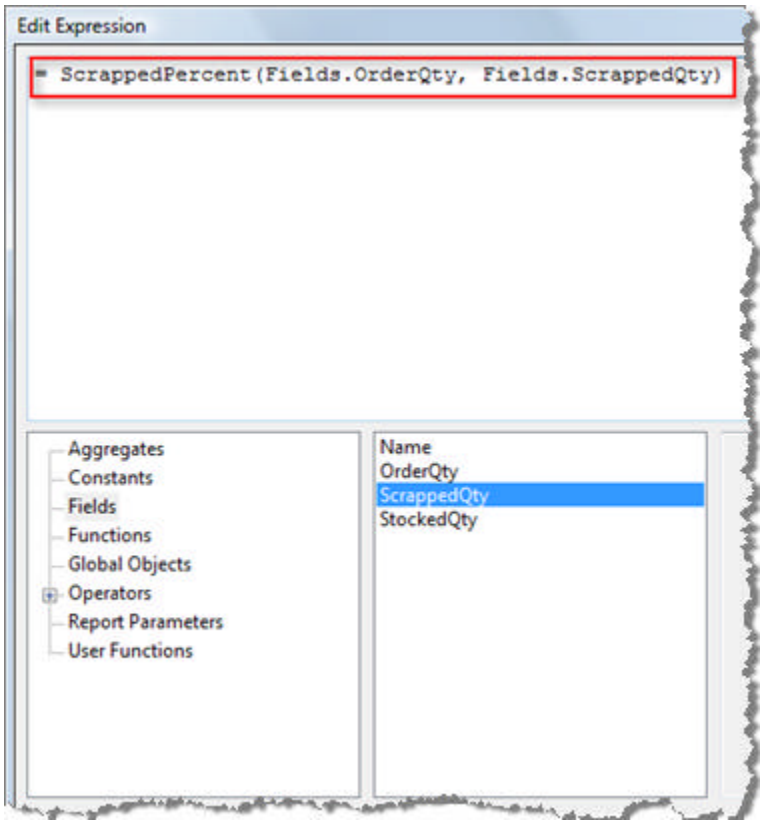
12. Back in the report designer, open the Telerik Reporting menu and select the **Data Explorer**. **Note**: if you don't see the Telerik Reporting menu, click on the report design surface - the menu will appear.
13. Drag the data fields to the Detail section of the report.
14. Add a **TextBox** report item to the right end of the Detail section. In the Property window for the TextBox item, set the **Name** property as "tbPercent". This TextBox will contain our expression.
15. Drag TextBox report items to the Page header section of the report for each data column and title them so that your design view looks something like the screenshot below.



16. Right-click the "tbPercent" **TextBox** and select **Expression...** from the context menu.
17. Click User Functions from the list on the lower left. Double-click the "Scrapped Percent" user function from the list on the right. Your expression should look something like the screenshot below:

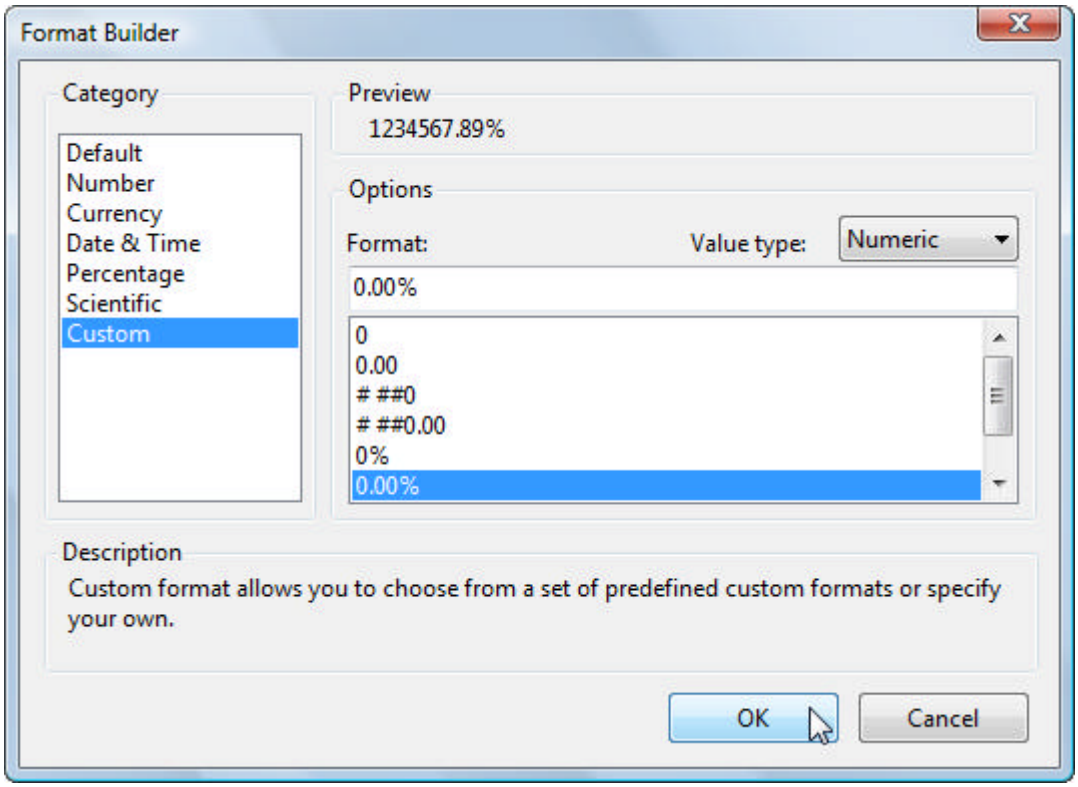


18. Click Fields from the list on the lower left.
19. Double-click "OrderQty" from the list on the right.
20. Manually enter a comma.
21. Double-click "ScrapQty" from the list on the right.

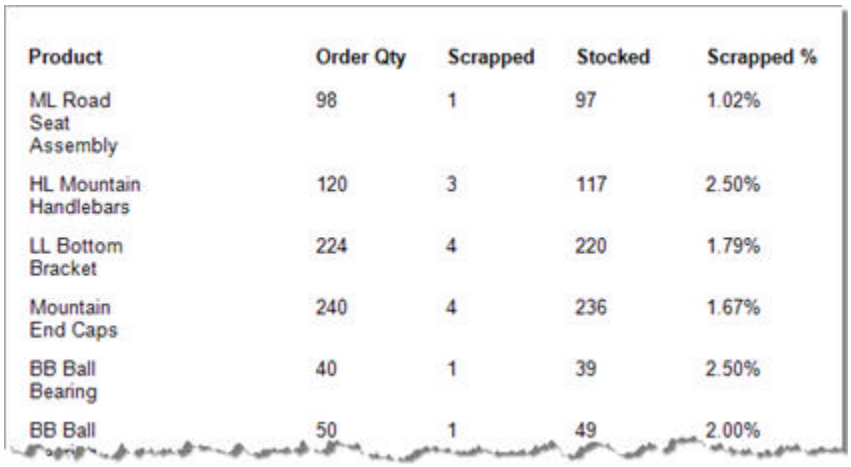


22. Click **OK** to close the Edit Expression dialog.
23. With "tbPercent" still selected, locate **Format** in the Properties window and click the ellipses to display the Format Builder dialog.

    Note: The Format Builder helps you create format strings that can be applied against your data. The dialog comes with a number of commonly used formats for numbers, currency, date/time, etc., or you can build your own custom format string.
24. In the **Category** list select "Custom". In the **Format** list select "0.00%"



25. Click the **Preview** tab to view the report. Notice the Scrapped % column data is properly formatted.



Product	Order Qty	Scrapped	Stocked	Scrapped %
ML Road Seat Assembly	98	1	97	1.02%
HL Mountain Handlebars	120	3	117	2.50%
LL Bottom Bracket	224	4	220	1.79%
Mountain End Caps	240	4	236	1.67%
BB Ball Bearing	40	1	39	2.50%
BB Ball	50	1	49	2.00%

Summary

This section reviewed the types of expressions available, how to create your own user defined functions and how to assign expressions at design-time

7 Grouping

Objectives

In this lesson you will learn how groups act on report data, and how to create groups at both design time and run-time.

Grouping Overview

Groups trigger breaks in the report based on changes in report data. You can have multiple levels of grouping nested within a report.

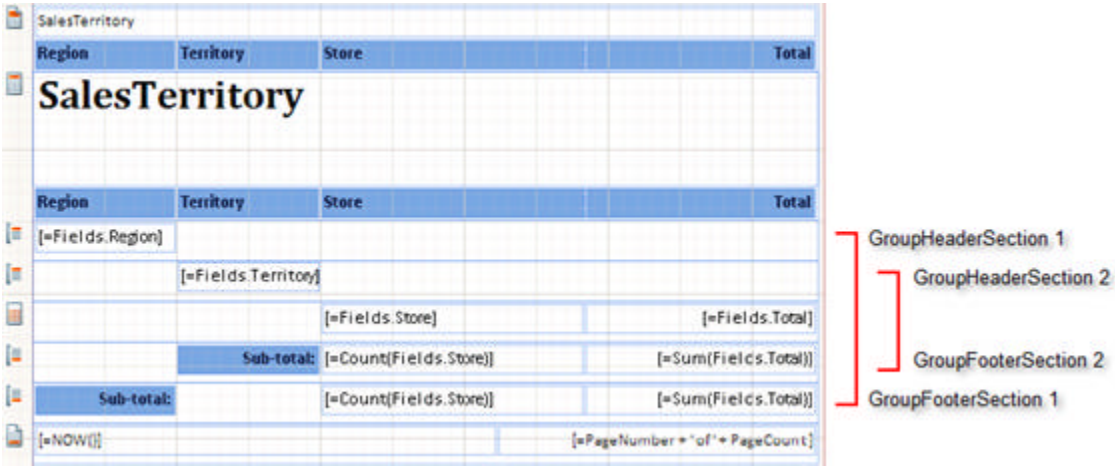
Below is a sales report example with columns for "Region", "Territory", "Stores" within territories and total sales for each store. Each region is grouped so that "US" has its own subtotal. Below that, each territory "Northeast" and "Southwest" has its own count of stores and subtotal for total store total sales.

SalesTerritory

Region	Territory	Store	Total
US			
	Northeast		
		Seventh Bike Store	$4,944.34
		Convenient Bike Shop	$5,135.49
		Curbside Universe	$1,668.61
		Weekend Tours	$16,891.08
		Retail Sales and Service	$8,998.81
	Sub-total:	5	$37,638.33
	Southwest		
		World Bike Discount Store	$556.20
		Wheel Gallery	$6,718.05
		Separate Parts Corporation	$13,012.40
		Area Bike Accessories	$15,105.89
		Fitness Toy Store	$45,187.51
		Grease and Oil Products Company	$1,159.98
		Bike Dealers Association	$51,056.50
	Sub-total:	7	$132,796.53
Sub-total:		36	$611,558.86

Each group has a **GroupHeaderSection** and **GroupFooterSection**. The GroupHeaderSection prints at the

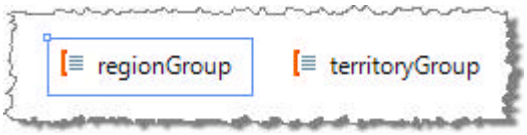
beginning of each group and typically has a name or description of the group. The GroupFooterSection prints after the detail data for the group and can contain summary data. You can use aggregate functions to create subtotals for each group. Each group can be filtered and sorted.



Grouping Properties

Group Properties

When you create a grouping level, Telerik Reporting adds a new **Telerik.Reporting.Group** component to the report's component tray area.



Each group component has these properties:

- **Name**: A descriptive name such as "regionGroup".
- **Filters:** This property determines which groups will display. This level of filtering occurs before filtering for the report as a whole occurs. The filters for the group are applied to the grouping data, not the report data itself.
- **GroupKeepTogether**: If this property is set to **FirstDetail,** then the group header and the first detail record are printed on the same page of output. A setting of **All** ensures that the entire group is printed on the same page of output. If there is not enough space on the current page, then rendering will skip to the top of the next page.
- **Grouping**: This property controls grouping criteria that determine where the grouping level breaks occur.
- **Sorting**: This property controls the sorting order for the grouping data, not the report data. This level of sorting occurs before the sorting for the report as a whole.

Group Header and Footer Section Properties

When you create a grouping level, Telerik Reporting adds a new group header section to the report. Select the header section to set the following properties:

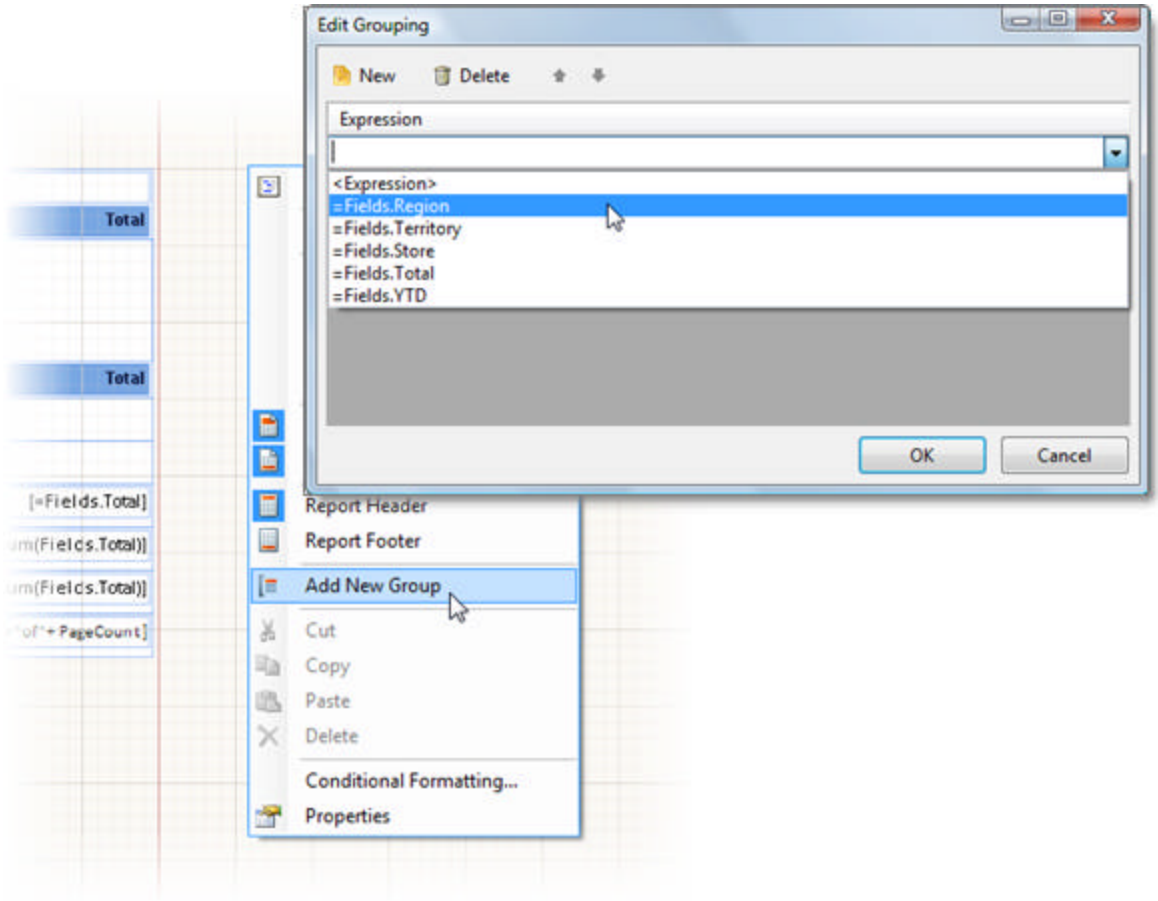
- **KeepTogether**: If **True** the entire section remains together on one printed page if possible. If there is not enough space on the current page, then rendering will skip to the top of the next page.
- **PageBreak**: Specifies where to put page breaks relative to this group, i.e. **None**, **Before**, **After**, or **BeforeAndAfter**.

- **PrintOnEveryPage**: If **True** the group header is repeated at the top of each page.
- **Visible**: Set to **False** to prevent the group from rendering.
- **ConditionalFormatting:** A collection containing rules and formatting to be applied to this report item if the rules succeed.

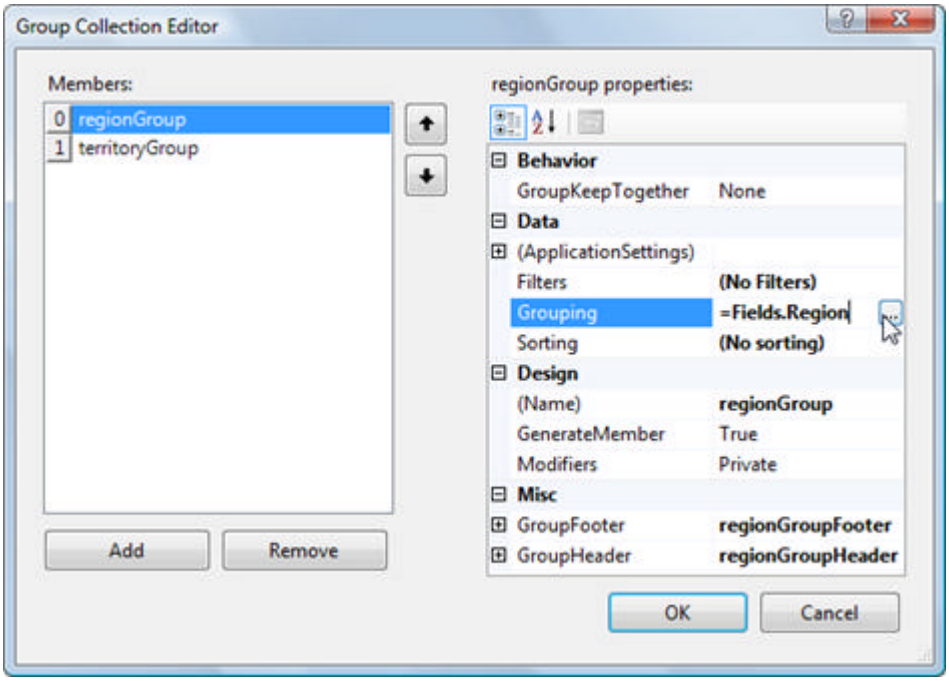
Creating Groups at Design Time

At design-time you can create report groupings by:

- Using the Design Data Layout page of the **Report Wizard**. Use this method when you need to set both the grouping and the general characteristics of the entire report quickly.
- Right-clicking the designer and selecting **Add New Group** from the context menu. Use this method to quickly add another grouping to an existing report. Clicking the menu option displays the **Edit Grouping** dialog where you can specify criteria for the grouping.



- Using the **Groups** collection of the report object. This method gives you greatest control over all the grouping related properties at one time.



Creating Groups at Run Time

Groups can be created on-the-fly in code. The general steps are:

- Create a **Group** object
- Create and assign a **Grouping** object to the group. The Grouping object includes the expression used by the group to know when the report break should occur.
- Create group header and footer sections.
- Create and assign report items to the group header and footer sections.

The example below takes a simple list of territories, stores within territories and total sales for each store. The query below defines the data:

[SQL] Territory Sales

```
SELECT TOP (50)
Sales.SalesTerritory.NAME AS Territory,
Sales.Store.NAME AS Store,
Sales.SalesOrderHeader.TotalDue AS Total
FROM Sales.SalesOrderHeader
INNER JOIN Sales.SalesTerritory
ON Sales.SalesOrderHeader.TerritoryID = Sales.SalesTerritory.TerritoryID
INNER JOIN Sales.Store
ON Sales.SalesOrderHeader.CustomerID = Sales.Store.CustomerID
INNER JOIN Sales.CustomerAddress
ON Sales.SalesOrderHeader.CustomerID = Sales.CustomerAddress.CustomerID
```

This example assumes that you already have a simple report with detail only against this data and want to group on "Territory". In the code example below we create groups, group header and footer sections and report items.

[C#] Creating Group, Sections and Report Items

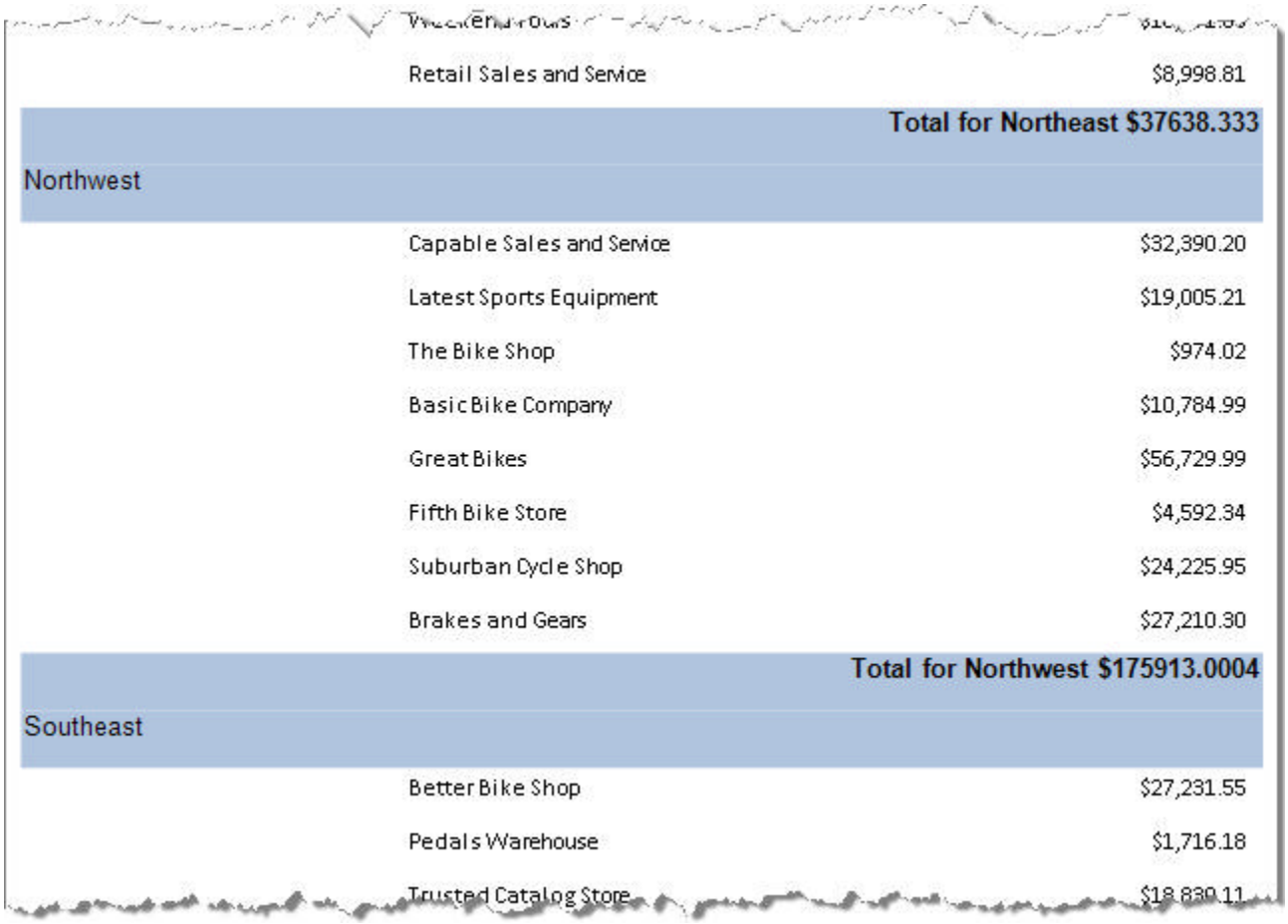
```
// requires Telerik.Reporting,
// Telerik.Reporting.Drawing and Telerik.Reporting.Data namespaces
// create the group,
// add the group expression that triggers the report break
// add the group to the report
Group group = new Group();
Grouping groupExpression = new Grouping("Territory");
group.Grouping.Add(groupExpression);
// create and size the group header, set to new color
group.GroupHeader = new GroupHeaderSection();
group.GroupHeader.Height = new Unit(0.3, UnitType.Inch);
group.GroupHeader.Style.BackgroundColor = Color.LightSteelBlue;
// create header report items, location defaults to the left
Telerik.Reporting.TextBox tbRegion = new Telerik.Reporting.TextBox();
tbRegion.Value = "=Territory";
tbRegion.Size = new SizeU(
  new Unit(1, UnitType.Inch),
  new Unit(0.25, UnitType.Inch));
group.GroupHeader.Items.Add(tbRegion);
// create and size the group footer, set to new color
group.GroupFooter = new GroupFooterSection();
group.GroupFooter.Height = new Unit(0.3, UnitType.Inch);
group.GroupFooter.Style.BackgroundColor = Color.LightSteelBlue;
// create footer report items,
// location set using Dock and Style.TextAlign properties
Telerik.Reporting.TextBox tbTotals = new Telerik.Reporting.TextBox();
tbTotals.Value = "=\"Total for \" + Fields.Territory + \" $\" + CStr(Sum(Fields.Total))";
tbTotals.Dock = DockStyle.Fill;
tbTotals.Style.TextAlign = HorizontalAlign.Right;
// use a currency format
tbTotals.Format = "{0:C2}";
tbTotals.Style.Font.Bold = true;
group.GroupFooter.Items.Add(tbTotals);
this.Groups.Add(group);
```

[VB] Creating Group, Sections and Report Items

```
' requires Telerik.Reporting,
' Telerik.Reporting.Drawing and Telerik.Reporting.Data namespaces
' create the group,
' add the group expression that triggers the report break
' add the group to the report
Dim group As New Group()
Dim groupExpression As New Grouping("Territory")
group.Grouping.Add(groupExpression)
' create and size the group header, set to new color
group.GroupHeader = New GroupHeaderSection()
group.GroupHeader.Height = New Unit(0.3, UnitType.Inch)
group.GroupHeader.Style.BackgroundColor = Color.LightSteelBlue
' create header report items, location defaults to the left
Dim tbRegion As New Telerik.Reporting.TextBox()
tbRegion.Value = "=Territory"
tbRegion.Size = New SizeU(New Unit(1, UnitType.Inch), New Unit(0.25, UnitType.Inch))
group.GroupHeader.Items.Add(tbRegion)
' create and size the group footer, set to new color
group.GroupFooter = New GroupFooterSection()
```

```
group.GroupFooter.Height = New Unit(0.3, UnitType.Inch)
group.GroupFooter.Style.BackgroundColor = Color.LightSteelBlue
' create footer report items,
' location set using Dock and Style.TextAlign properties
Dim tbTotals As New Telerik.Reporting.TextBox()
tbTotals.Value = "=""Total for "" + Fields.Territory + "" $"" + CStr(Sum(Fields.Total))"
tbTotals.Dock = DockStyle.Fill
tbTotals.Style.TextAlign = HorizontalAlign.Right
' use a currency format
tbTotals.Format = "{0:C2}"
tbTotals.Style.Font.Bold = True
group.GroupFooter.Items.Add(tbTotals)
Me.Groups.Add(group)
```

The report ends up looking like the sample screenshot below:



	Retail Sales and Service	$8,998.81
		Total for Northeast $37638.333
Northwest		
	Capable Sales and Service	$32,390.20
	Latest Sports Equipment	$19,005.21
	The Bike Shop	$974.02
	Basic Bike Company	$10,784.99
	Great Bikes	$56,729.99
	Fifth Bike Store	$4,592.34
	Suburban Cycle Shop	$24,225.95
	Brakes and Gears	$27,210.30
		Total for Northwest $175913.0004
Southeast		
	Better Bike Shop	$27,231.55
	Pedals Warehouse	$1,716.18
	Trusted Catalog Store	$18,839.11

Summary

This lesson demonstrated the basic usage of groups and grouping related properties, how to create groups using the designer and how to create groups using code only.

8 Filtering

Objectives

- Understand how **Filter Rules** act on report data and groups of data.
- Learn how to create and apply filter rules at design-time and in code.
- Consider the performance implications for sorting operations.

Filter Behavior

Filters limit the number of records in a report based on one or more filter rules. If the rule conditions are met, the record is included in the report. Filters are defined using the the **Edit Filter Dialog** and can also be created in code.

Report Without Filter

Let's start with a simple set of data that contains Product categories, sub categories and products from the AdventureWorks database. The report is grouped by Category and SubCategory. If you output the data without filtering it looks something like the example below. All the categories, sub categories and all products are listed. Each category contains a number of sub categories and each sub category has one or more products.

Category	SubCategory	Product
Accessories		
	Bike Racks	
		Hitch Rack - 4-Bike
	Bike Stands	
		All-Purpose Bike Stand
	Bottles and Cages	
		Water Bottle - 30 oz.
		Mountain Bottle Cage
		Road Bottle Cage
	Cleaners	
		Bike Wash - Dissolver
	Fenders	
		Fender Set - Mountain

Report Level Filtering

We can filter against the report data so that if a detail record meets the rule criteria, the record is included in the report output. The example below shows records where the Product name starts with the character "C".

Category	SubCategory	Product
Accessories		
	Locks	
		Cable Lock
Clothing		
	Vests	
		Classic Vest, S
		Classic Vest, M
		Classic Vest, L
Components		
	Chains	
		Chain

Group Level Filtering

We can also limit the groups in the report. In the example below the Category group is filtered where only categories starting with the character "C" are included in the output.

Category	SubCategory	Product
Clothing		
	Bib-Shorts	
		Men's Bib-Shorts, S
		Men's Bib-Shorts, M
		Men's Bib-Shorts, L
	Caps	
		AWC Logo Cap
	Gloves	
		Half-Finger Gloves, S
		Half-Finger Gloves, M
		Half-Finger Gloves, L
		Full-Finger Gloves, S
		Full-Finger Gloves, M
		Full-Finger Gloves, L
Components		
	Bottom Brackets	
		LL Bottom Bracket
		ML Bottom Bracket
		HL Bottom Bracket
	Brakes	
		Rear Brakes
		Front Brakes

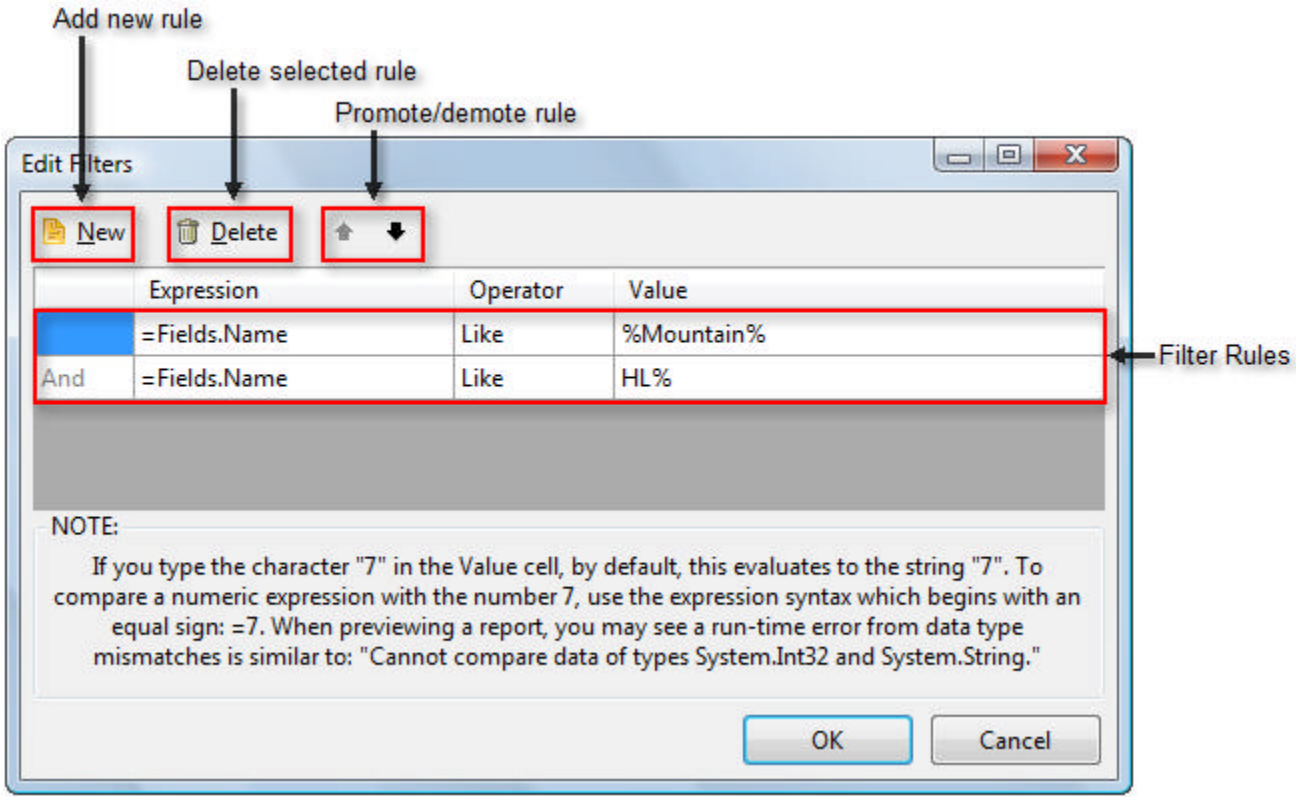
Combined Report and Group Filtering

You can apply multiple filter rules and also multiple filters at one time. In this next example we retain the Category group level filter and add a report level filter that looks for products that contain "Mountain" AND products that start with "HL". Both group and report level filters produce output something like the example below.

Category	SubCategory	Product
Components		
	Handlebars	
		HL Mountain Handlebars
	Mountain Frames	
		HL Mountain Frame - Silver, 42
		HL Mountain Frame - Silver, 44
		HL Mountain Frame - Silver, 48
		HL Mountain Frame - Silver, 46
		HL Mountain Frame - Black, 42
		HL Mountain Frame - Black, 44
		HL Mountain Frame - Black, 48
		HL Mountain Frame - Black, 46
		HL Mountain Frame - Black, 38
		HL Mountain Frame - Silver, 38

Using the Edit Filters Dialog

To create a filter at design time, use the **Edit Filters** dialog. With either the report or a group component selected in the designer, locate the **Filter** property in the Property window. Click the **Filter** property ellipses to invoke the Edit Filters dialog. The dialog has a tool bar that lets you add and delete rules and to rearrange rules in the list.



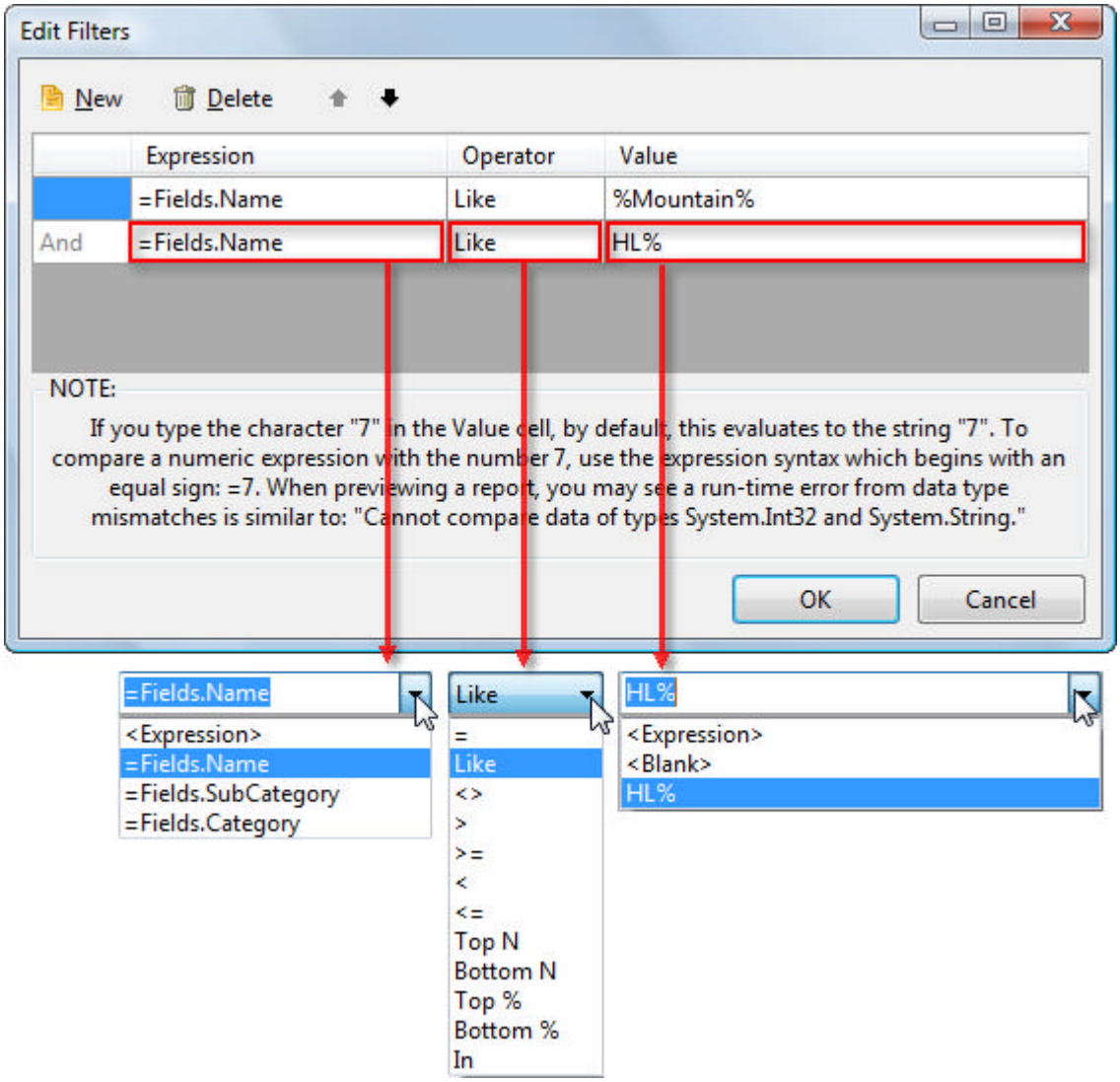
Each filter rule is made up of an **Expression**, **Operator** and **Value**.

- **Expression** can be a data field from the **Fields** collection or defined using the Edit Expression dialog.
- **Operator** can be simple comparison operators but can also include **Like** and additional operators to include the top and bottom number of records and the top and bottom percentage of records. The full list of operators are: >, <, >=, <=, =, Like, Top N, Bottom N, Top %, Bottom % and IN.

  Use **Like** with the "%" wildcard to represent any number of characters.

  Top N and **Bottom N** return a certain number of records. If **Expression** is blank the filter returns the exact number of records specified by **Value.** For example "Top N 5" returns exactly five records. If **Expression** contains a field or expression the filter returns the top number for that expression. This filter may return more than the exact count specified in **Value.** For example "ProductCategory Top N 2" might return twenty records containing ProductCategory "Accesories" and "Apparel".

  Top % and **Bottom %** work in a similar way. If **Expression** is blank the filter returns the percentage of total records specified in **Value.** If the report has an initial 1000 records and the filter is "Top % 5", 50 records are returned. Likewise, if **Expression** contains a field or expression the filter returns the top percentage records that satisfy the expression. For example if the top two percent country codes in a StateProvince table are "AS", "AU" and "CA", the filter might return twenty records containing only those country codes.
- **Value** can be a literal value, "<blank>" or an expression defined in the Edit Expression dialog.



Lab: Creating a Filter Using the Edit Filter Dialog

The following walk-through demonstrates creating a report with multiple levels of grouping. The report will be filtered at report and group levels.

1. Create a new report class.
2. When the Report Wizard displays the Welcome to the Telerik Report Wizard page, click **Next**.
3. In the Report Choice page select **New Report** and click **Next**.
4. In the Choose Data Source page select **Create New Data Source**
5. In the Choose Data Source Type page click **Next**.
6. In the Choose Data Connection page click **New Connection**.
7. In the **Add Connection** dialog:
    1. Set the Data Source to be Microsoft SQL Server
    2. Set the server name to be <your server>\SQLEXPRESS
    3. Use Windows Authentication and select AdventureWorks as the database name.

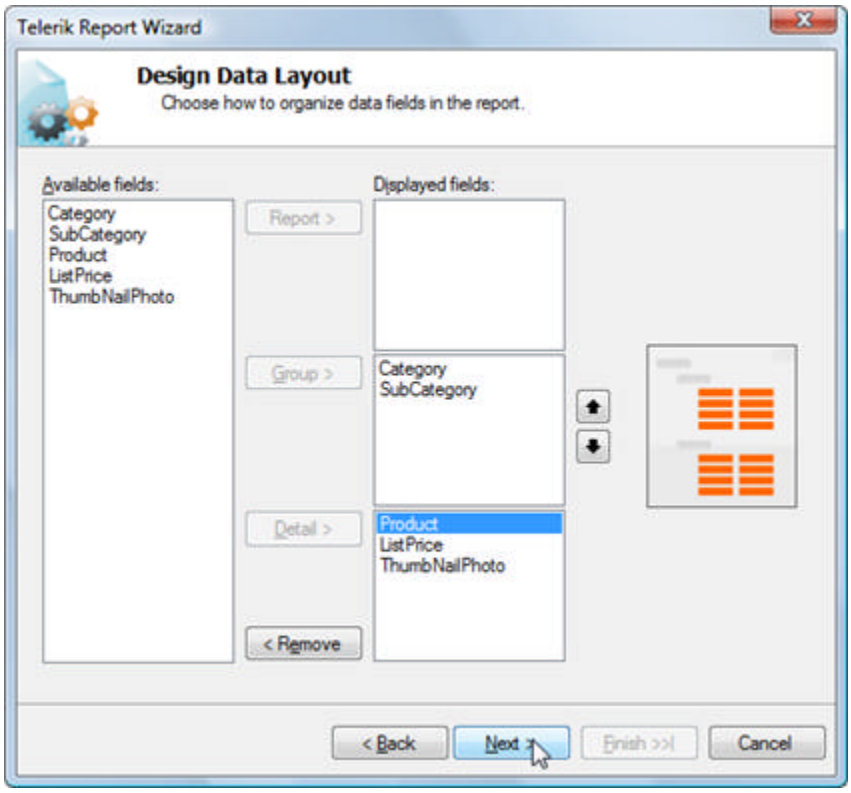
4. Click **OK** to close the Add Connection dialog.

8. In the Choose Data Connection page, select the database connection you just created and click **Next**.
9. In the Choose your Database Objects page, paste the query below and click **Next**:

[SQL] Product Categories Query

```
SELECT
Production.ProductCategory.NAME AS Category,
Production.ProductSubcategory.NAME AS SubCategory,
Production.Product.NAME AS Product,
Production.Product.ListPrice,
Production.ProductPhoto.ThumbNailPhoto
FROM Production.Product
INNER JOIN Production.ProductSubcategory
ON Production.Product.ProductSubcategoryID =
Production.ProductSubcategory.ProductSubcategoryID
INNER JOIN Production.ProductCategory
ON Production.ProductSubcategory.ProductCategoryID =
Production.ProductCategory.ProductCategoryID
INNER JOIN Production.ProductProductPhoto
ON Production.Product.ProductID = Production.ProductProductPhoto.ProductID
INNER JOIN Production.ProductPhoto
ON Production.ProductProductPhoto.ProductPhotoID = Production.ProductPhoto.ProductPhotoII
```

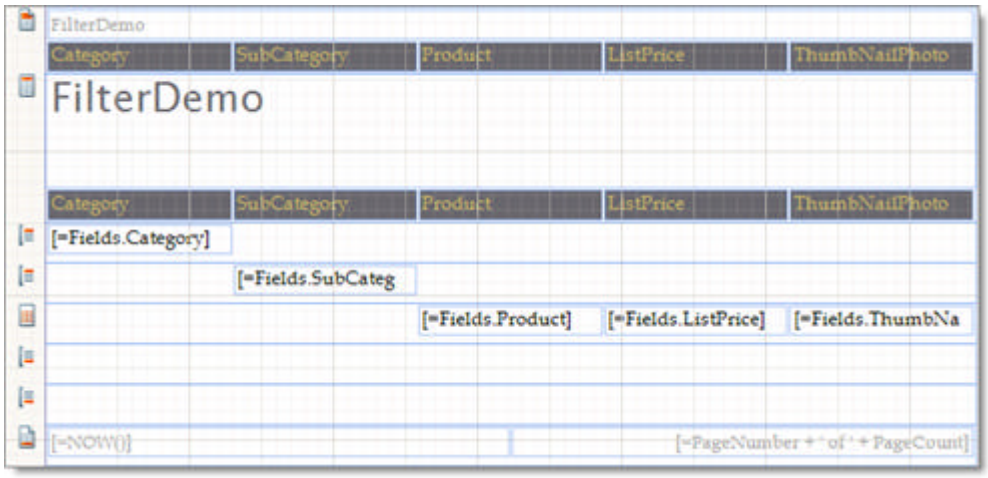
10. On the Select Report Type page of the wizard, select **Standard** and click **Next**.
11. In the Design Data Layout page, populate the Group Displayed Fields with "Category" and "SubCategory". Populate Detail with "Product", "ListPrice" and "ThumbNailPhoto". Click the **Next** button.



12. In the Choose Report Layout page of the wizard, click the **Next** button.

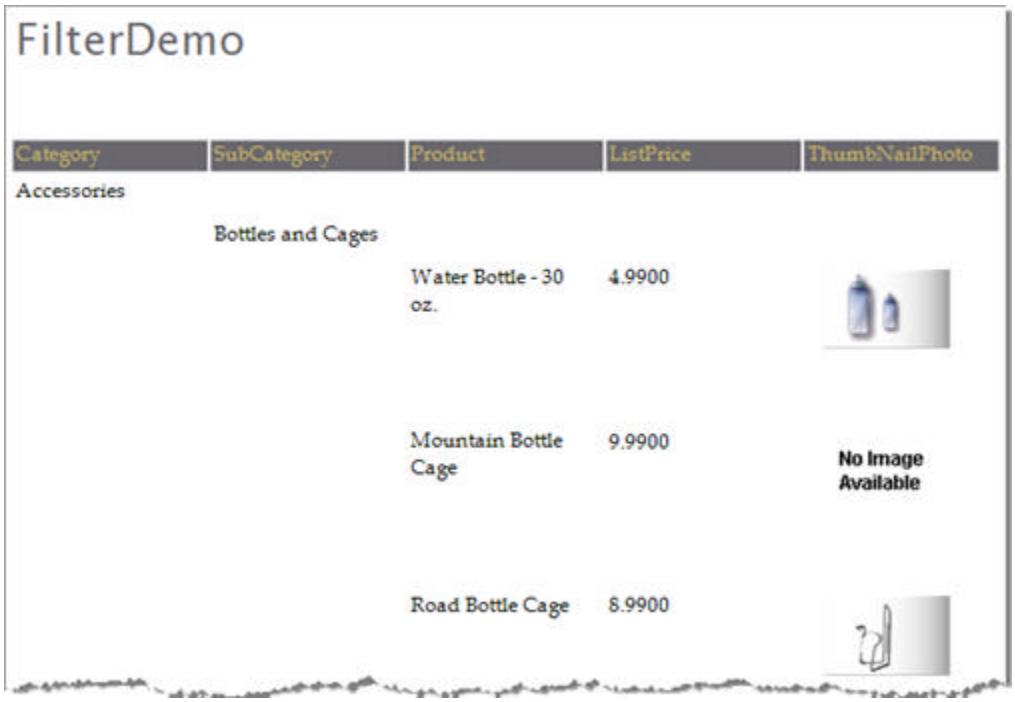
13. In the Choose Report Style page of the wizard, select Apex and click **Next**.
14. In the Completing the Wizard page of the wizard, click the **Finish** button to close the wizard.

    The designer should now look something like the example below.



15. Delete the TextBox on the right of the report that contains the ThumbNailPhoto binding expression.
16. Drag a PictureBox report item to replace the TextBox.
17. Right-click the PictureBox report item and click **Expression...** from the context menu.
18. In the Edit Expression dialog, click **Fields** in the list on the lower left. This will display the database fields currently available in the list on the right. Double-click "ThumbNailPhoto" to add the field name to the item binding expression. Click **OK** to close the Edit Expression dialog.
19. Click in the report design area outside the sections to select the Report object.
20. In the Properties window, locate the Filters property and click the ellipses. The Edit Filters dialog will display.
21. Click the **New** button to create a new filter rule. Make the rule settings:

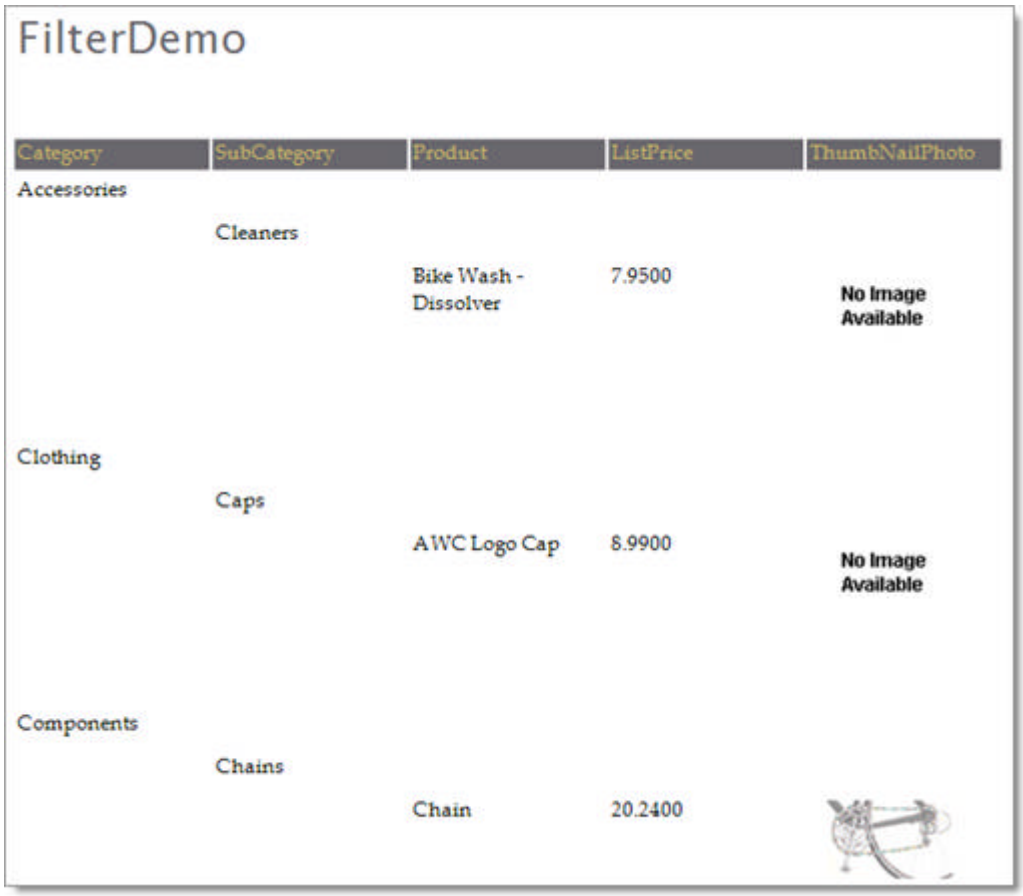
    Expression: "=Fields.ListPrice"
    Operator: "<"
    Value: "100"
22. Click **OK** to close the Edit Filters dialog.
23. Click the **Preview** tab to view the report so far. The output should show products with a list price of less than $100, and will show images if they are available.



24. Now locate the "subCategoryGroup" component in the component tray and select it with the mouse.
25. In the Properties window, locate the Filters property and click the ellipses. The Edit Filters dialog will display.
26. Click the **New** button to create a new filter rule. Make the rule settings:

    Expression: "=Fields.SubCategory"
    Operator: "Like"
    Value: "C%"
27. Click **OK** to close the Edit Filters dialog.
28. Click the **Preview** button to see the final report.

    Now you will see data where the sub category starts with a "C" character and where the list price is less than $100.



Lab: Creating a Filter in Code

Filters can also be added to the report or group Filters collection using code. This walk-through uses the previous project from "Creating a Filter" as a starting point and recreates the exact same filters:

1. First delete the filters for the subCcategoryGroup component and the filters for the report.
2. Add **Telerik.Reporting.Data** to your "using" (C#) or Imports (VB) namespace.
3. In the Report object constructor add the following code:

   [C#] Adding Report and Group Filters

   ```
   // Filter the report data for records where ListPrice is less than 100
   Filter reportFilter = new Filter("=Fields.ListPrice", FilterOperator.LessThan, "100");
   Report.Filters.Add(reportFilter);
   //  Filter the sub category group for records where the sub category starts with "C"
   Filter subCategoryFilter = new Filter("=Fields.SubCategory", FilterOperator.Like, "C%");
   subCategoryGroup.Filters.Add(subCategoryFilter);
   ```

   [VB] Adding Report and Group Filters

   ```
   ' Filter the report data for records where ListPrice is less than 100
   Dim reportFilter As New Filter("=Fields.ListPrice", FilterOperator.LessThan, "100")
   Report.Filters.Add(reportFilter)
   ' Filter the sub category group for records where the sub category starts with "C"
   Dim subCategoryFilter As New Filter("=Fields.SubCategory", FilterOperator.[Like], "C%")
   subCategoryGroup.Filters.Add(subCategoryFilter)
   ```

4. In the designer, click the **Preview** button to see the report.

FilterDemo

Category	SubCategory	Product	ListPrice	ThumbNailPhoto
Accessories				
	Cleaners			
		Bike Wash - Dissolver	7.9500	No Image Available
Clothing				
	Caps			
		AWC Logo Cap	8.9900	No Image Available
Components				
	Chains			
		Chain	20.2400	

Performance Considerations

It's convenient to create complex selections right within Telerik Reports, but it doesn't always perform as well as SQL running on a dedicated server machine. One approach is not inherently better than another. Is there a simple rule to know when sorts and selects should be performed at the database vs. performed in the reporting engine?

Not really, but there are some guidelines to help you make the best judgement. Consider the tradeoff between user selections applied at the database vs. selections applied to data at the report level. With large amounts of database server data you need to retrieve the smallest usable set of data to reduce network traffic, use resources effectively and to render reports quickly.

- Selections applied at the database level are effective when the data does not need to be re-queried. This can be effective when only a narrow range of data is required. Additional trips to the database for different views of the same data increase the network traffic cost and increase the time needed to render the report.
- When user selections are applied at the report level the data is cached in-memory. This can be effective when different views of the same data are required and the data is not too large.

A combination of these techniques may be the most appropriate solution. For example, consider sales data that includes sales person and sales date information. The selection at the database could query for a given date range and selections for particular sales people could be applied at the report level. The individual requirements of the report help determine the best solution.

Sorting also raises similar performance issues.

Summary

This section explored how filter rules work when they are applied to report data vs. filtering groups. You created a filter with multiple filter rules using the Edit Filter dialog and assigned the filter to the report data. You also recreated this same filter in code. This section also looked at performance considerations for filtering using the mechanisms supplied by Telerik Reporting vs using the database engine on a server.

9 Sorting

Objectives

- Understand how sorting acts on report data and against groups.
- Learn how to sort groupings and detail data using the **Edit Sorting** dialog.
- Use the Sorting collection to programmatically sort groups and detail report data.

Sorting Behavior

Sorting can be performed at the group and report level through the **Sorting** property. Each sort has an **Expression** and a **Direction**. Expression can be a data field from the **Fields** collection or an expression defined using the Edit Expression Dialog. Direction can be either "Asc" (ascending) or "Desc" (descending). Sorting order is defined using the **Sorting** property **Edit Sorting** dialog at design time, or at runtime you can use the report or group's **Sorting** collection directly.

Un-sorted Report

Reports display data in the order that it comes from the data source. If the data source uses a query with an "order by" clause, then the data will be sorted in the report output. If the data source sends data in the same order that records were input to the database, likewise, the data will display as un-sorted. In this next example, a vendor listing displays state, city and vendor name in no particular order.

Vendor List

State	City	Vendor
CA	Santa Cruz	Litware, Inc.
WA	Lynnwood	Advanced Bicycles
CA	Altadena	Allenson Cycles
CA	West Covina	American Bicycles and Wheels

Report Detail Sorted

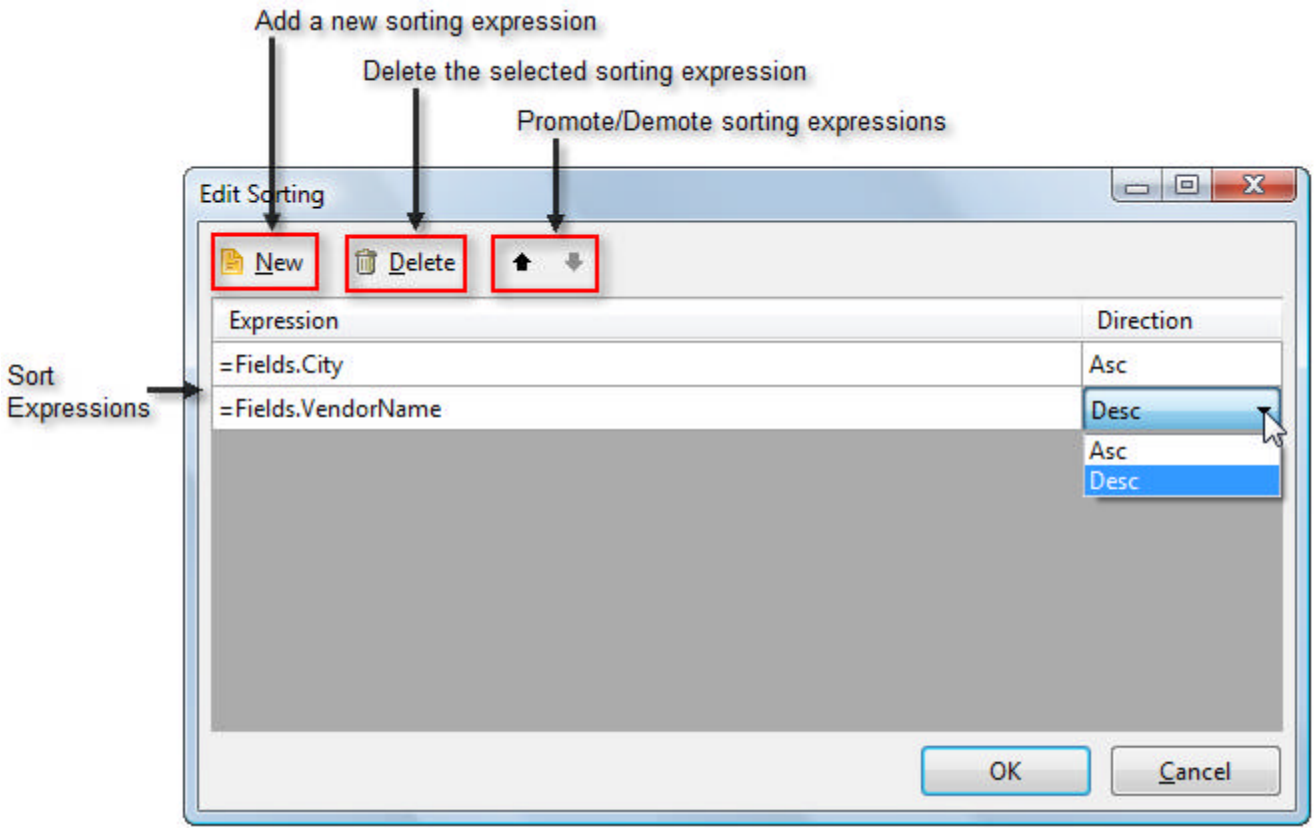
We can set the sort order first on state, then cities with states and finally vendors within cities. Both State and City are sorted in ascending order while vendor is sorted in descending order.

Group and Detail Sorted

If we group the data by state, we can sort the states and still sort the detail at the same time. Sorting for groups controls the order of grouping data, not the report detail data. Group level sorting occurs before the sorting for the report as a whole.

Using the Edit Sorting Dialog

To define sorting criteria at design time, use the **Edit Sorting** dialog. With either the report or a group component selected in the designer, locate the **Sort** property in the Property window. Click the **Sort** property ellipses to invoke the **Sort Filters** dialog. The dialog has a tool bar that lets you add sorting expressions and to rearrange expressions in the list.



Each sorting expression is made up of an **Expression** and a **Direction**. **Expression** can be a data field from the **Fields** collection or defined using the Edit Expression dialog. Direction can be **Desc** or **Asc**.

Lab: Sorting at Design-Time

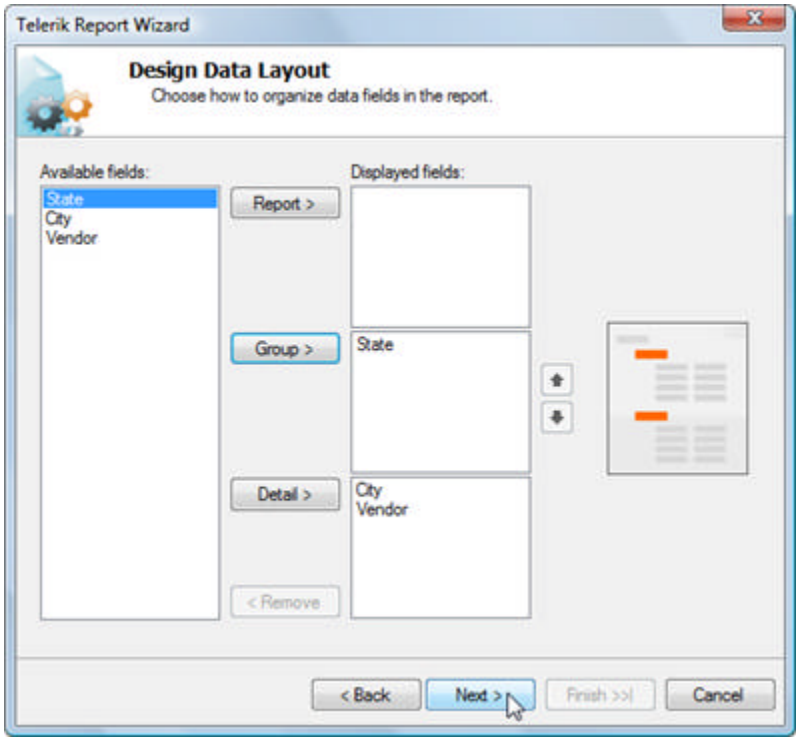
The following walk-through demonstrates sorting both group and detail report data.

1. Create a new report class.
2. When the Report Wizard displays the Welcome to the Telerik Report Wizard page, click **Next**.
3. In the Report Choice page select **New Report** and click **Next**.
4. In the Choose Data Source page select **Create New Data Source**
5. In the Choose Data Source Type page click **Next**.
6. In the Choose Data Connection page click **New Connection**.
7. In the **Add Connection** dialog:
    1. Set the Data Source to be Microsoft SQL Server
    2. Set the server name to be <your server>\SQLEXPRESS
    3. Use Windows Authentication and select AdventureWorks as the database name.
    4. Click **OK** to close the **Add Connection** dialog.
8. In the Choose Data Connection page, select the database connection you just created and click **Next**.
9. In the Choose your Database Objects page, paste the query below and click **Next**:

[SQL] Vendor Listing

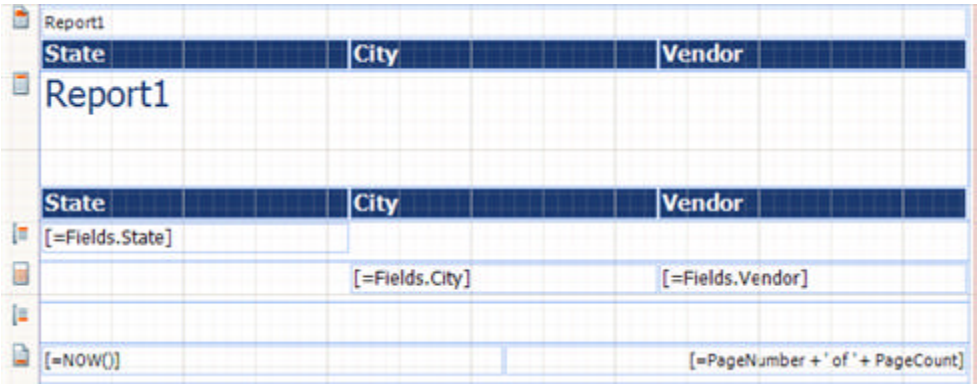
```
SELECT
 Person.StateProvince.StateProvinceCode AS State,
 Person.Address.City,
 Purchasing.Vendor.NAME AS Vendor
FROM
 Person.StateProvince
INNER JOIN Person.Address
ON Person.StateProvince.StateProvinceID = Person.Address.StateProvinceID
INNER JOIN Purchasing.VendorAddress
ON Person.Address.AddressID = Purchasing.VendorAddress.AddressID
INNER JOIN Purchasing.Vendor
ON Purchasing.VendorAddress.VendorID = Purchasing.Vendor.VendorID
```

10. On the Select Report Type page of the wizard, select **Standard** and click **Next**.
11. In the Design Data Layout page, populate the Group Displayed Fields with "State". Populate Detail with "City" and "Vendor". Click the **Next** button.



12. In the Choose Report Layout page of the wizard, click the **Next** button.
13. In the Choose Report Style page of the wizard, select "Corporate" and click **Next**.
14. In the Completing the Wizard page of the wizard, click the **Finish** button to close the wizard.

The designer should now look something like the example below.



15. Double click the TextBox "Report1" in the report header section. Enter "Vendor Listing".
16. Click in the report design area outside the sections to select the Report object.
17. In the Properties window, locate the **Sorting** property and click the ellipses. The Edit Sorting dialog will display.
18. Click the **New** button to create a new sort expression. Make the expression settings:

    Expression: "=Fields.City"
    Direction: "Asc"
19. Click the **New** button to create a second sort expression. Make the expression settings:

    Expression: "=Fields.Vendor"
    Direction: "Asc"
20. Click **OK** to close the Edit Sorting dialog.
21. Click the **Preview** tab to view the report so far. The output should show vendors listed within state and city. The city and vendor are sorted in ascending order with vendor sorted within city. Coincidentally, the first two instances of "State" appear to be in ascending order, but this happens to be the default order. Next we will sort the State grouping to change that.

22. Now locate the "stateGroup" component in the component tray and select it with the mouse.

23. In the Properties window, locate the **Sorting** property and click the ellipses. The Edit Sorting dialog will display.
24. Click the **New** button to create a new sort expression. Make the expression settings:

    Expression: "=Fields.State"
    Direction: "Desc"
25. Click **OK** to close the Edit Sorting dialog.
26. Click the **Preview** button to see the final report.

    Now you will the grouped states listed in descending order. City and Vendor retain their ascending sorts.

Vendor Listing

State	City	Vendor
WY		
	Novato	Indiana Bicycle Center
WA		
	Anacortes	Electronic Bike Co.
	Anacortes	Northern Bike Travel
	Anacortes	Speed Corporation
	Ballard	Reliance Fitness, Inc.
	Bellevue	Fitness Association

Lab: Handling Sorting Programmatically

Sorting can also be applied to the report or group using code. This walk-through uses the previous project from "Sorting at Design Time" as a starting point and recreates the exact same sorting:

1. First delete the sort expressions for the stateGroup component and the sort expressions for the report.
2. Add **Telerik.Reporting.Data** to your "uses" (C#) or Imports (VB) namespace.
3. In the Report object constructor add the following code:

   [C#] Adding Report and Group Sorting

```
// Sort the report data by city, then vendor within city
this.Sorting.Add(
   new Sorting("City", Telerik.Reporting.Data.SortDirection.Asc));
this.Sorting.Add(
   new Sorting("Vendor", Telerik.Reporting.Data.SortDirection.Asc));
// Sort the "State" group in descending order
stateGroup.Sorting.Add(
   new Sorting("State", Telerik.Reporting.Data.SortDirection.Desc))
```

   [VB] Adding Report and Group Sorting

```
' Sort the report data by city, then vendor within city
Me.Sorting.Add(New Sorting("City", Telerik.Reporting.Data.SortDirection.Asc))
Me.Sorting.Add(New Sorting("Vendor", Telerik.Reporting.Data.SortDirection.Asc))
' Sort the "State" group in descending order
```

```
stateGroup.Sorting.Add(New Sorting("State", Telerik.Reporting.Data.SortDirection.Desc))
```

4. In the designer, click the **Preview** button to see the report.

Summary

This section explored how sorting works against report detail data and how you can also sort groups. You learned how to apply sorting at design time and programmatically.

10 Parameterized Reports

Objectives

In this section you will learn how parameters work with expressions to amplify the power of Telerik Reporting. You will see how parameters can make your reporting infrastructure easier to maintain and more flexible for your users. This section also explains the new features of parameters including the automatic parameter UI and Cascading parameters. When you're done creating your parameterized reports, we will create a Windows application that displays the report, and will automatically display parameter prompts for the user.

Parameter Basics

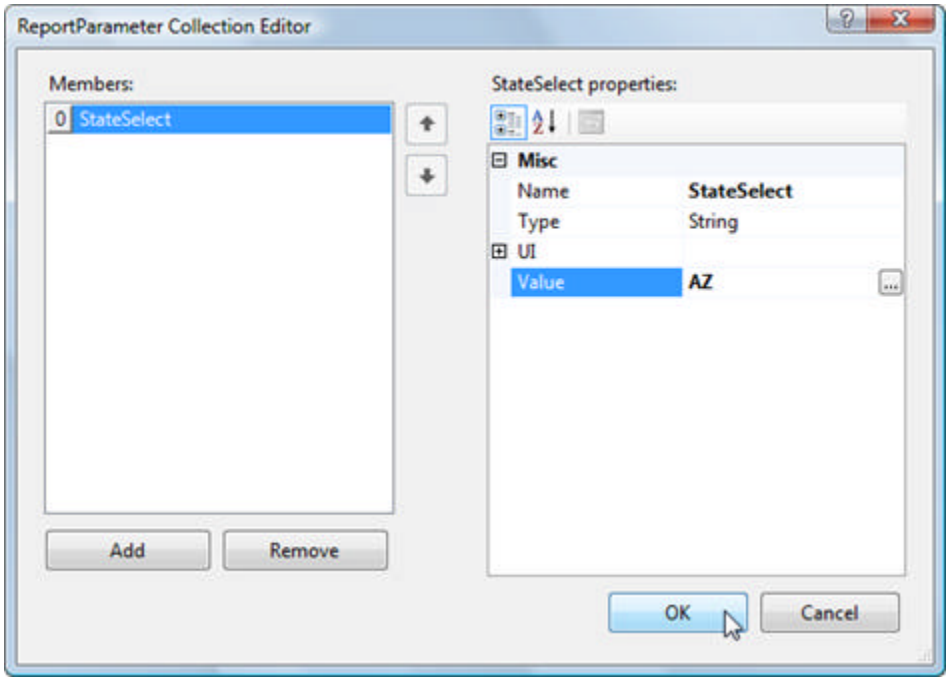
Report Parameters allow users to tailor their own data. Parameters are particularly powerful when coupled with expressions and can be used with filtering, sorting and grouping. Parameters help make your reporting infrastructure flexible and generic. Your customers will be able to get more done before needing maintenance help or before you are forced to modify similar but not-quite-the-same reports ("Say, now can you filter that by customer?").

Telerik Reporting supports **Boolean**, **DateTime**, **Integer**, **Float** and **String**, or collections of any of these types. Define your parameters at design time using the **ReportParameter Collection Editor** or at runtime using the **ReportParameters** collection. Users can also enter parameter values directly into the automatically created UI of the Report Viewer.

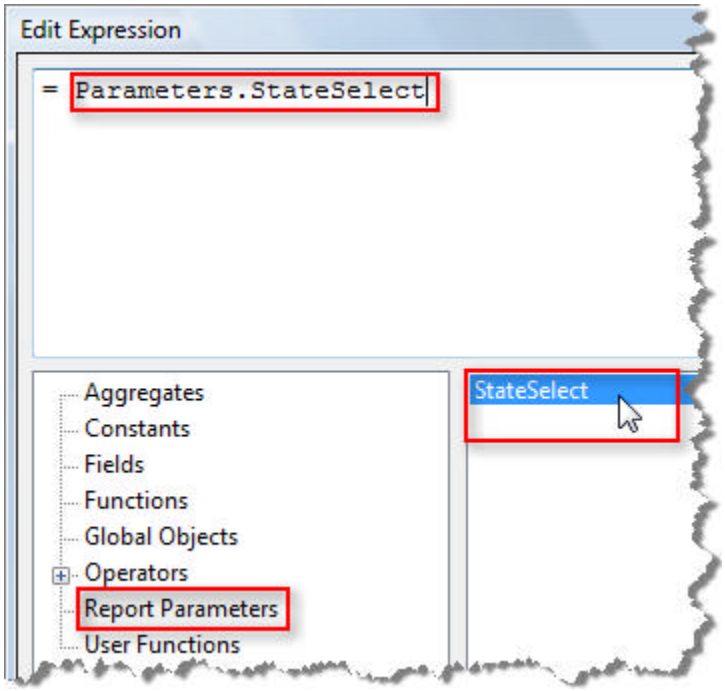
The **SubReport** item also has a **Parameters** collection that can be used for master/detail or other kinds of aggregated reports.

Defining Parameters at Design-Time

To make any use of parameters you will need to consume parameters within expressions. The first step is to create a parameter so that later it will be visible in the **Expression Editor**. Use the report's **ReportParameter** property to invoke the **ReportParameter Collection Editor** dialog. In the dialog you define a parameter by clicking the **Add** button, defining a **Name** and **Type** of the parameter and optionally assigning a **Value**. The Value can be hard-coded or set to be the result of an expression (click the Value property ellipses to invoke the Edit Expression dialog.



When you click **OK** to close this dialog, the parameter becomes available in any expression throughout the report.



Using Parameters with Expressions

The following walk-through demonstrates creating a parameter and using it to select from a simple vendor listing. The basic report is based off the following query that lists Vendor, State, City, Address and Postal Code.

Before parameters are introduced, the query produces a simple listing:

VendorListing

Vendor	State	City	Address	Postal Code
Litware, Inc.	CA	Santa Cruz	4405 Balboa Court	95062
Advanced Bicycles	WA	Lynnwood	7995 Edwards Ave.	98036
Allenson Cycles	CA	Altadena	4659 Montoya	91001
American Bicycles and Wheels	CA	West Covina	1667 Warren Street	91791
American Bikes	CA	Torrance	7179 Montana	90505
Anderson's Custom Bikes	CA	Burbank	9 Guadalupe Dr.	91502
Proseware, Inc.	OR	Lebanon	50 Big Canyon Road	97355
Aurora Bike Center	WA	Port Orchard	65 Park Glen Court	98366

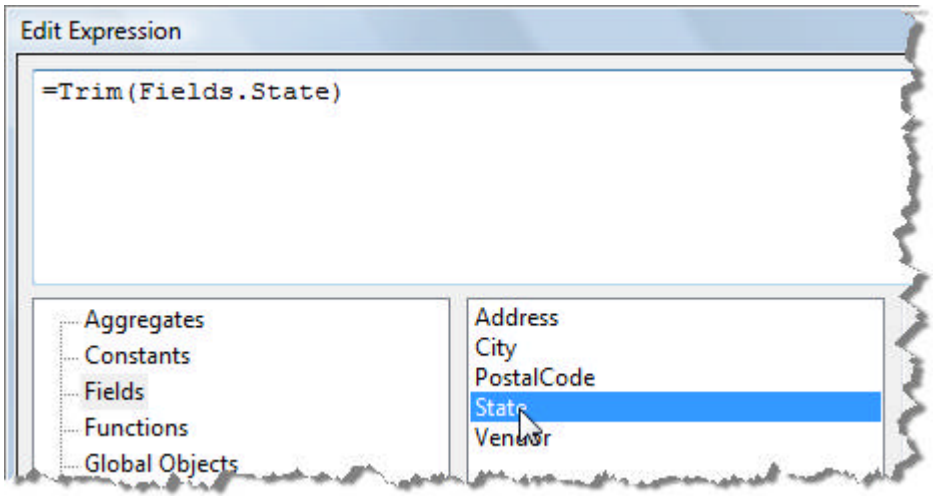
1. Create a new report class using the AdventureWorks database and the query below. Include all the fields from the query in your report.

   [SQL] Vendor Listing Query

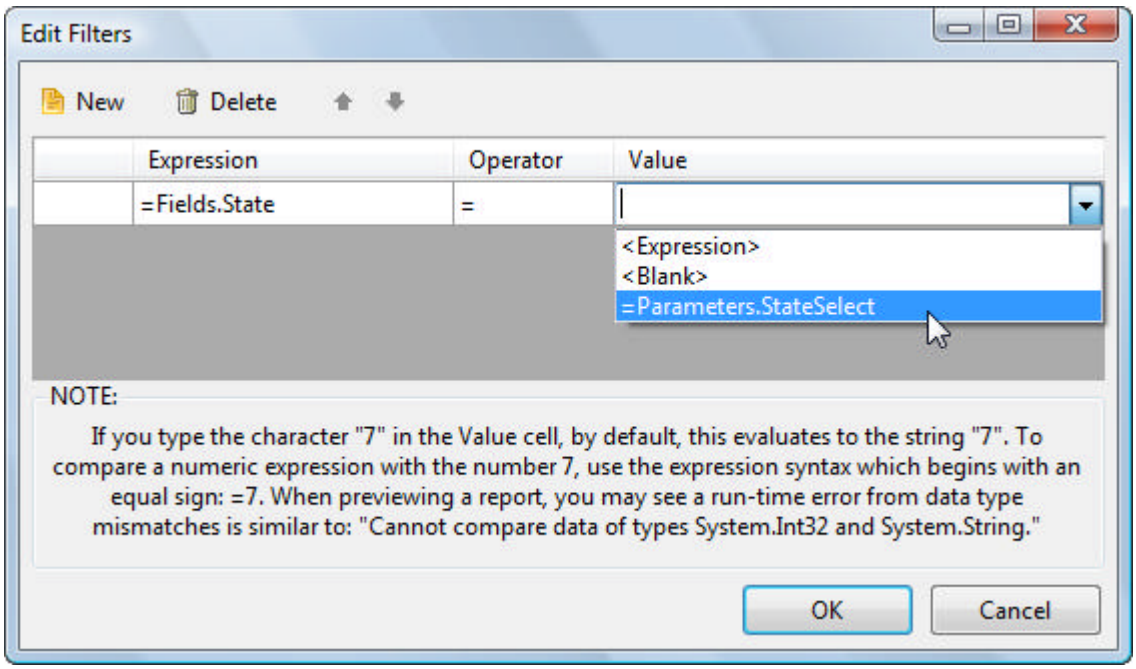
```
SELECT
 Purchasing.Vendor.NAME AS Vendor,
 Person.StateProvince.StateProvinceCode AS State,
 Person.Address.City,
 Person.Address.AddressLine1 AS Address,
 Person.Address.PostalCode
FROM Purchasing.VendorAddress
INNER JOIN Purchasing.Vendor
ON Purchasing.VendorAddress.VendorID = Purchasing.Vendor.VendorID
INNER JOIN Person.Address
ON Purchasing.VendorAddress.AddressID = Person.Address.AddressID
INNER JOIN Person.StateProvince
ON Person.Address.StateProvinceID = Person.StateProvince.StateProvinceID
```

2. Using the designer, select the report and locate the **ReportParameters** property in the Property Window.
3. Click the ReportParameters ellipses to display the **ReportParameter Collection Editor** dialog.
4. Click the **Add** button to create a parameter.
5. Set the parameter **Name** as "StateSelect", **Type** as String and **Value** as "AZ".
6. Click **OK** to close the **ReportParameter Collection Editor** dialog.
7. Locate the report **Filters** property in the Property Window and click the ellipses to open the **Edit Filters** dialog.
8. Click the **New** button to create a new filter rule.
9. In the **Expression** column of the filter rule select "<Expression>" from the drop down list. The **Edit Expression** dialog will display.
10. In the lower left hand list of the Edit Expression dialog, select **Functions**.

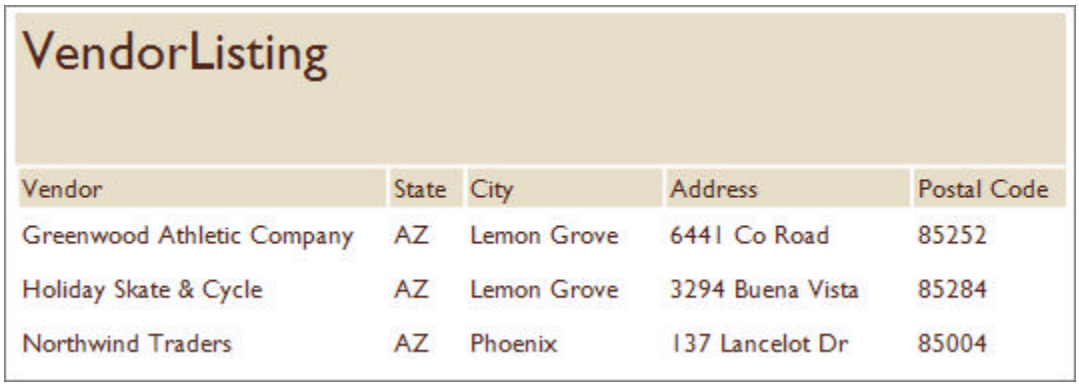
11. From the list of functions double-click **System.String.Trim()** so that it is added to the expression.
12. In the Edit Expression lower left hand list, select **Fields**. From the list of fields, double-click "**State**" so that it is added to the expression. The expression should now look like the screenshot:



13. Click **OK** to accept the new expression and close the dialog.
14. Leave the **Operator** column of the filter rule as "=".
15. Choose "=Parameters.StateSelect from the **Value** drop down list.



16. Click **OK** to close the **Edit Filters** dialog.
17. In the designer, click the **Preview** tab. The report will now only show records where the State is "AZ" as defined by our "StateFilter" parameter.



VendorListing

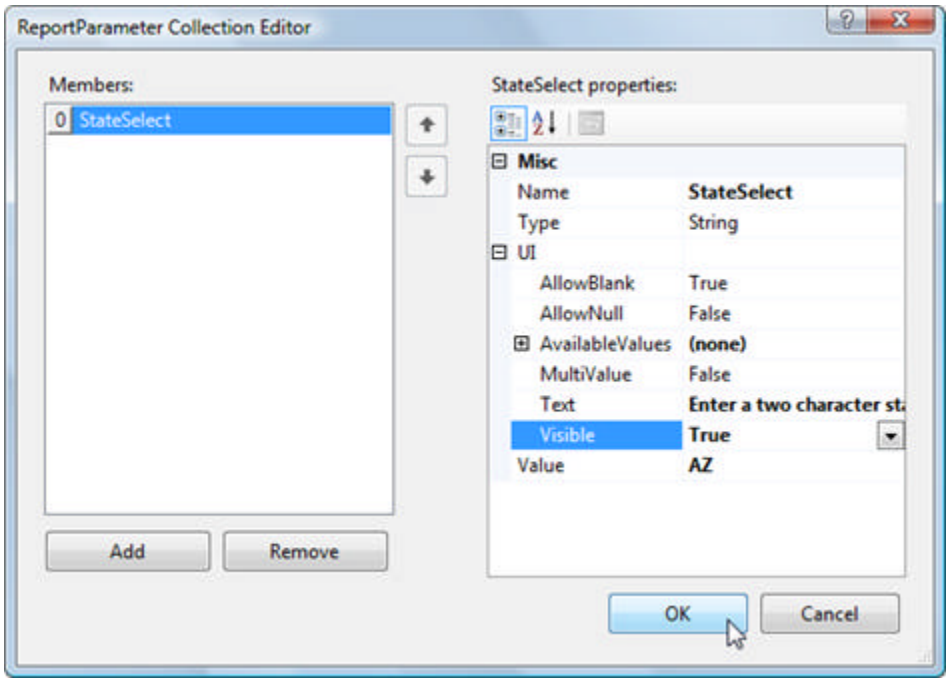
Vendor	State	City	Address	Postal Code
Greenwood Athletic Company	AZ	Lemon Grove	6441 Co Road	85252
Holiday Skate & Cycle	AZ	Lemon Grove	3294 Buena Vista	85284
Northwind Traders	AZ	Phoenix	137 Lancelot Dr	85004

Automatic Parameter UI

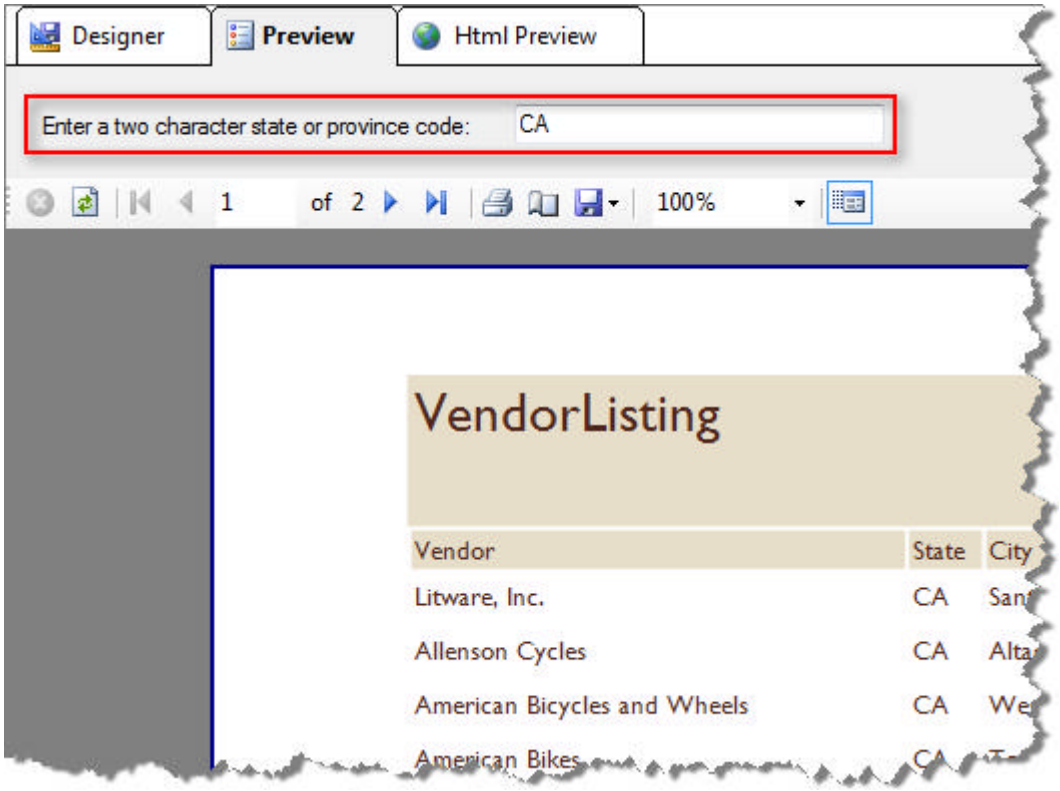
Instead of changing a ReportParameter Value at design time or coding it, user input can be collected for you automatically. Based on your configuration of the report parameters, a user interface is generated within the designer Preview tab or in a report viewer.

Supplying Prompts for Simple Values

In the **ReportParameter Collection Editor** dialog you may have noticed a **UI** property for the ReportParameter. The user interface is generated based on **UI** property values. In the example below the **UI.Text** is defined as "Enter a two character state or province code". This will be displayed in the report viewer as a prompt if **UI.Visible** is set to True.

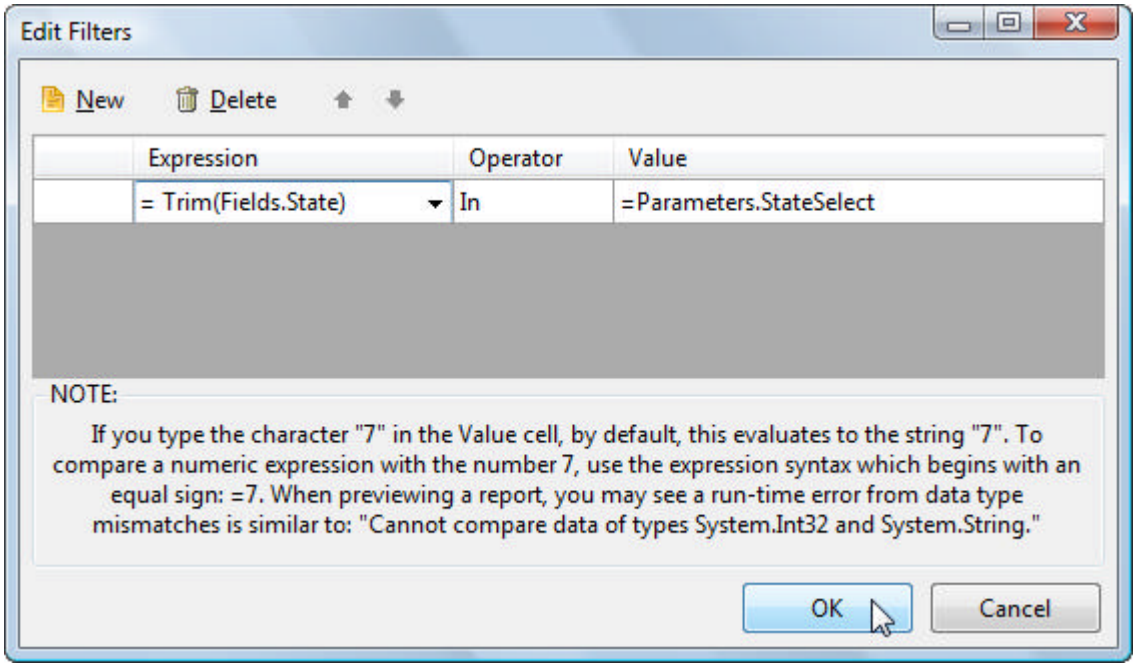


In the Preview tab or in one of the Report Viewers, you will now have a prompt displayed in a format appropriate for the parameter. The user can enter new values and hit the preview button to get new sorted versions of the report.

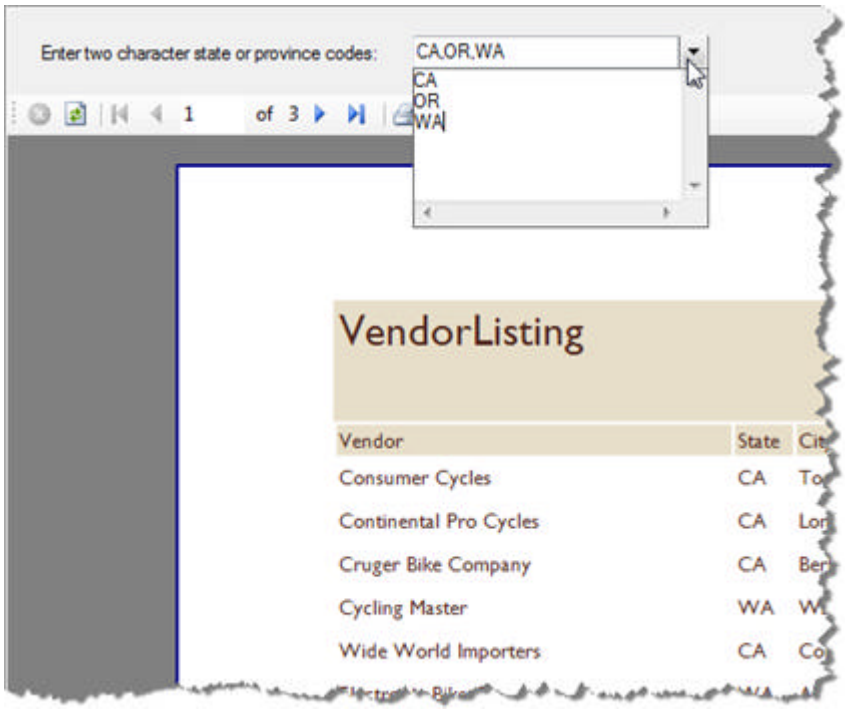


Multiple Values

With one minor tweak you can filter against a list of values. Set the **UI.MultiValue** property to True and the parameter will accept a comma-delimited list of values. You also need to change the expression that employs the parameter so that the "In" comparison operator is used.



Now the report preview shows a new editor that lets you edit multiple values, and the report output lists multiple states that match:



To pre-populate the the list of values in code, assign an **ArrayList** to the **Value** property:

[C#] Populating the Value

```
ArrayList list = new ArrayList();
list.Add("CA");
list.Add("OR");
```

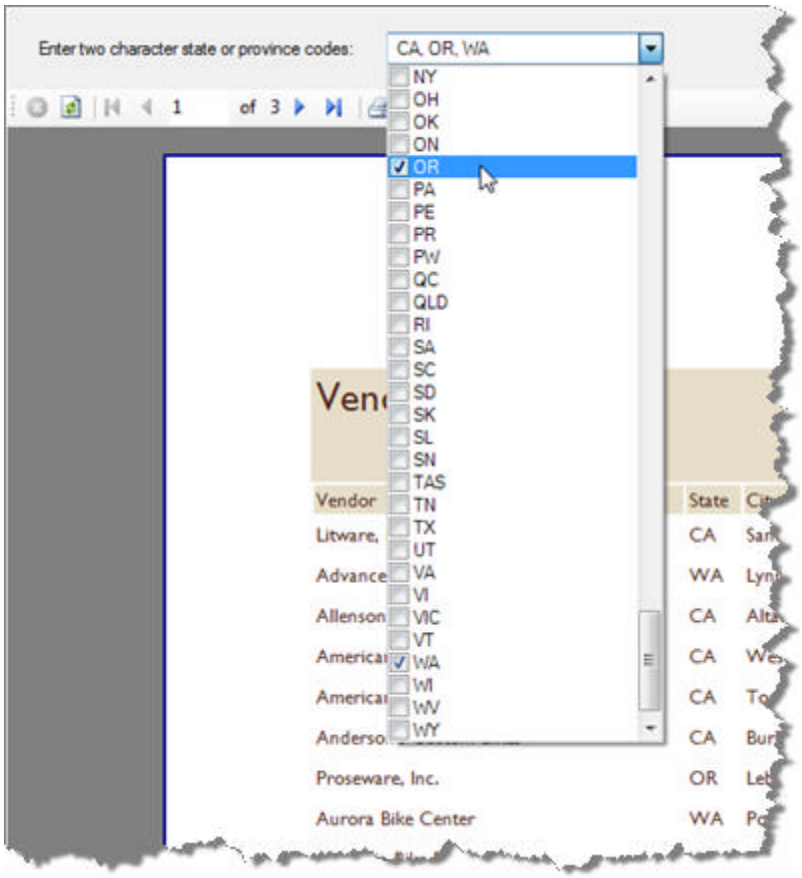
```
list.Add("WA");
this.ReportParameters[0].Value = list;
```

[VB] Populating the Value

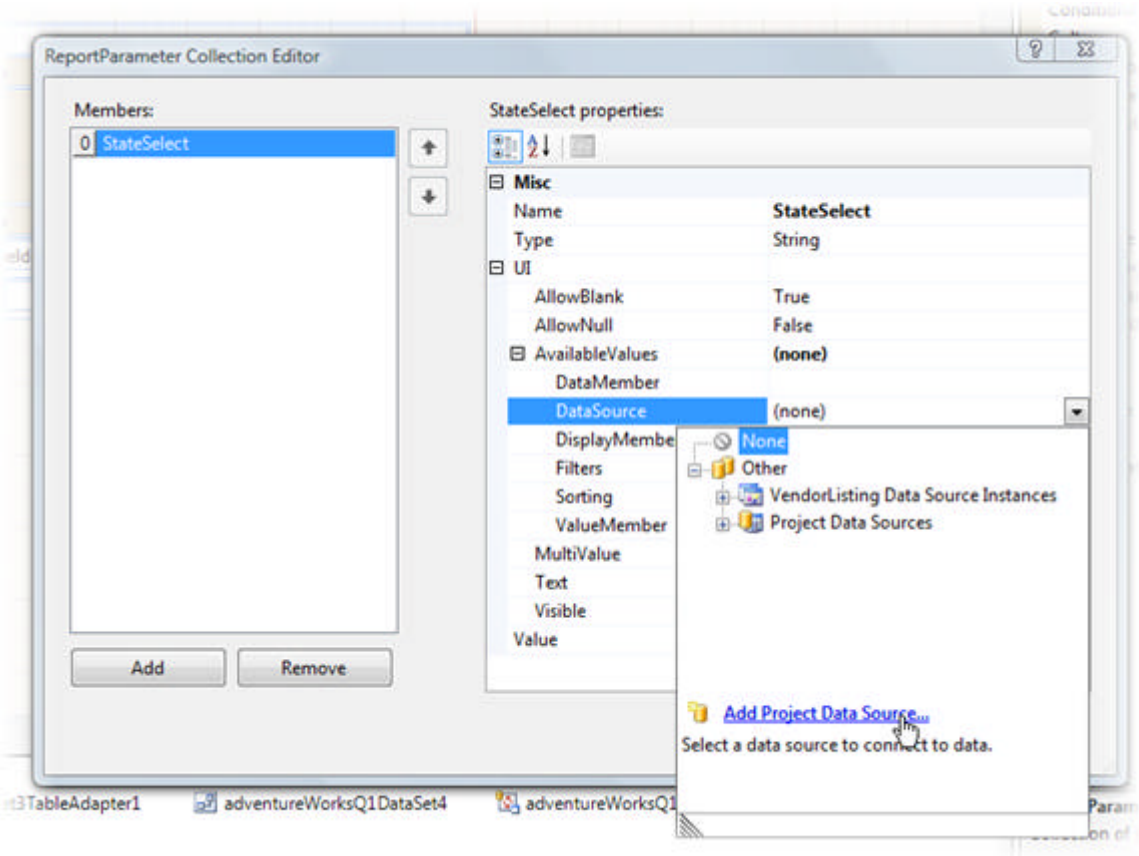
```
Dim list As New ArrayList()
list.Add("CA")
list.Add("OR")
list.Add("WA")
Me.ReportParameters(0).Value = list
```

Getting Available Values from a Database

Of course, having the user enter "AZ" or "WA" is not the optimum solution. We want to supply a list of values from a database so the user doesn't have to know the valid values and can't make a mistake. If the values in the database change, the report prompt should change automatically.



To get a list of values from the database, assign the ReportParameter **DataSource** property. You can use the ReportParameter DataSource property editor **Add Project Data Source** option to create a new datasource from scratch.

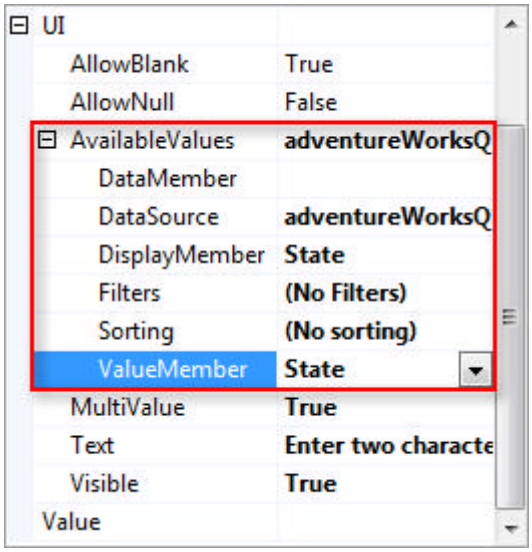


When you define your data source, you can create a simple query against your master data to form a list, for example:

[SQL] Query to List State and Province Codes

```
SELECT RTRIM(LTRIM(StateProvinceCode)) AS State
FROM Person.StateProvince
```

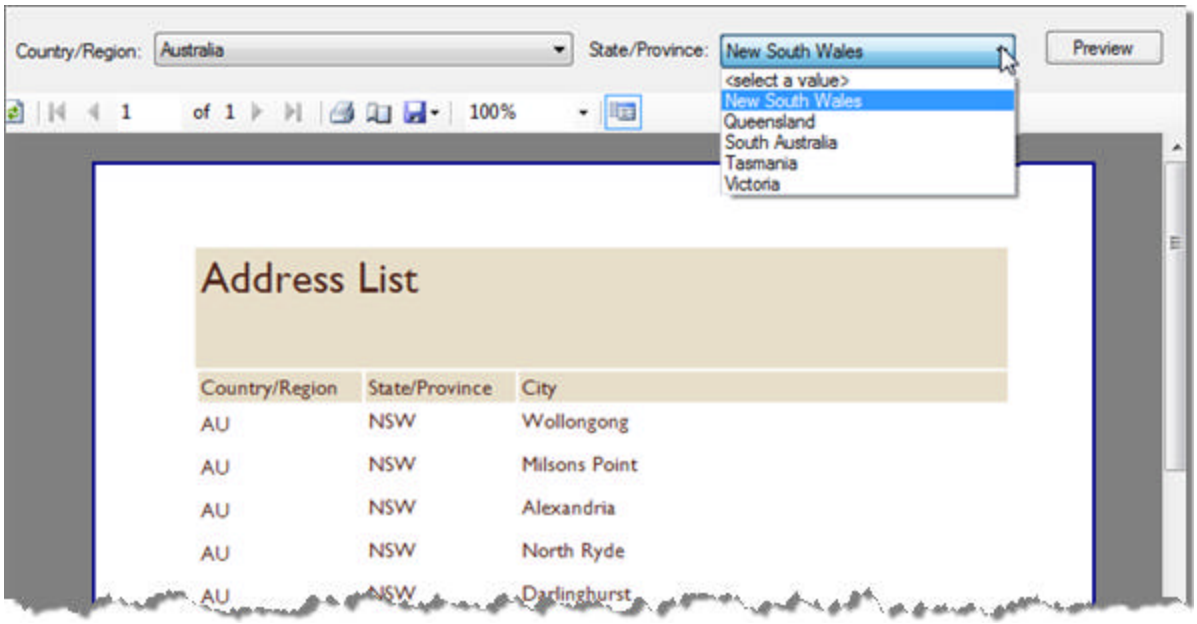
After you set the DataSource property for the parameter, set the **DisplayMember** property to a column that will show in the drop down list. Set the **ValueMember** to the column that should be retrieved when the user selects an item from the list. In this case, we are using the "State" code for both value and display members. **Note**: If your DataSource contains multiple tables, you also need to select a **DataMember**.



Cascading Parameters

Cascading parameters allow you to have dependencies between parameters so that large amounts of data can be progressively narrowed down. This technique is typically used against data that forms a hierarchy, i.e. country/region/city, header/detail/line items, warehouse aisle/bay/bin, etc.

The user experience is that the user chooses from the top-most level of data first, for example "Australia" for country. Then the user has access to regions and can choose "New South Wales". When the user clicks the Preview button, they can see a list of all cities within the New South Wales region.



1. To recreate the example report shown in the screenshot above, start by creating a new report class based on the SQL below.

   [SQL] List Country/Region, State/Province and City

```
SELECT DISTINCT
 Person.StateProvince.CountryRegionCode,
 Person.StateProvince.StateProvinceCode,
```

```
 Person.Address.City
FROM Person.Address
INNER JOIN Person.StateProvince
ON Person.Address.StateProvinceID = Person.StateProvince.StateProvinceID
```

The resulting report output without parameters and filtering will be an un-ordered listing of countries, state/provinces and cities.

Address List

Country/Region	State/Province	City
US	NM	Rio Rancho
US	IL	Norridge
US	WI	Mosinee
DE	NW	München
US	IL	West Chicago

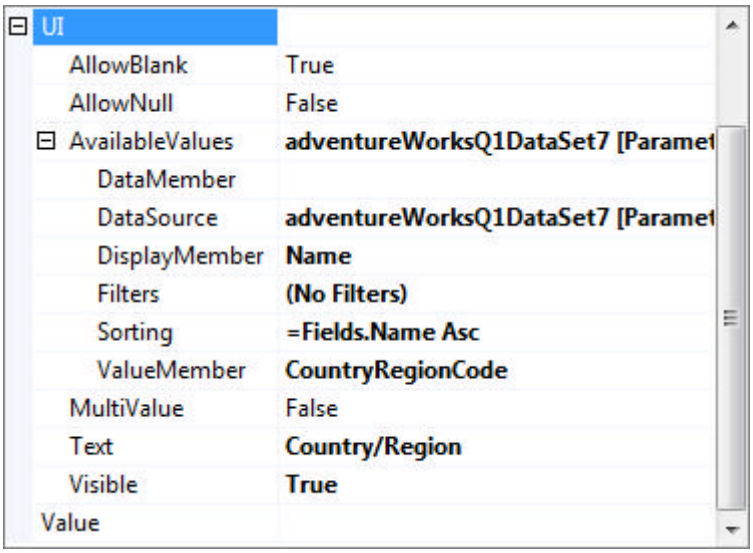
2. In the Property Window, locate the **ReportParameters** property and click the ellipses.
3. In the **ReportParam Collection Editor** dialog, add a new parameter.
   1. Set the Name property to "CountryRegionParam".
   2. Open the **UI** sub-property, set the **Text** property to "Country/Region" and **Visible** to True.
   3. Open the **UI.AvailableValues** property and locate **DataSource.** Drop down the **DataSource** property and click the **Add Project Data Source** option.
   4. Create the new data source using the SQL below. The SQL produces a list of country names along with two character codes.

      [SQL] List Country/Region

      ```
      SELECT NAME, CountryRegionCode
      FROM Person.CountryRegion
      ```

   5. With the DataSource set, drop down the **UI.AvailableValues.DisplayMember** property and select "Name". Likewise, drop down the list for **UI.AvailableValues.ValueMember** and select "CountryRegionCode".
   6. Set the **UI.AvailableValues.Sorting** property to "=Fields.Name Asc".

      The parameter properties should now look like this screenshot:

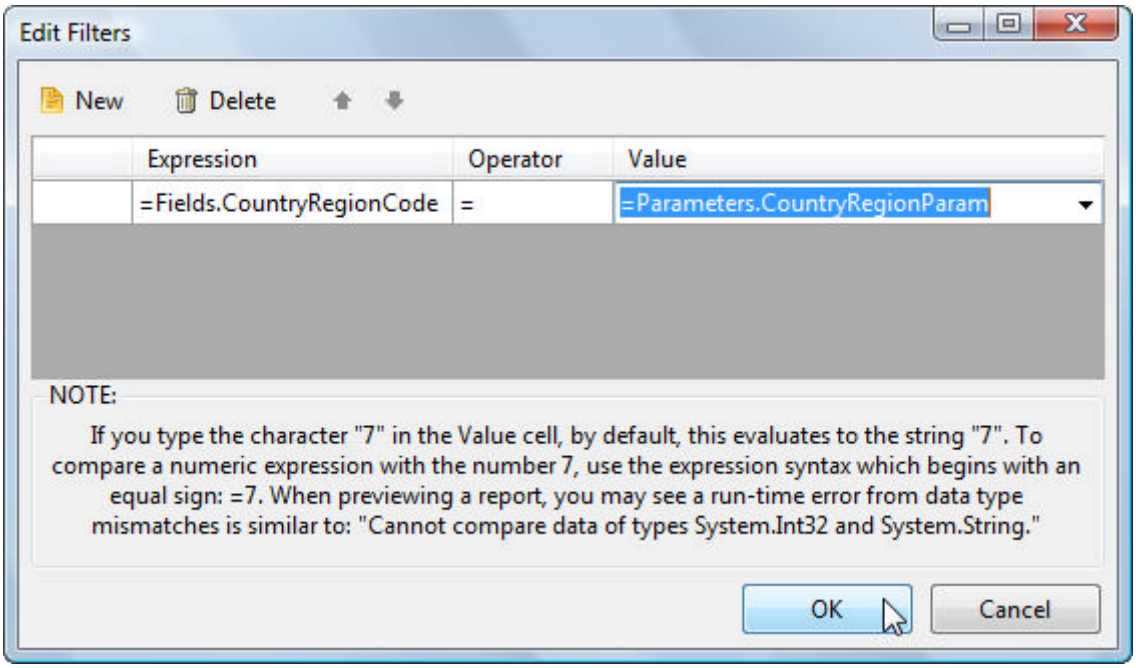


4. In the ReportParam Collection Editor dialog, add a second parameter.
   1. Open the **UI** sub-property, set the **Text** property to "Country/Region" and **Visible** to True.
   2. Open the **UI.AvailableValues** property and locate **DataSource.** Drop down the **DataSource** property and click the **Add Project Data Source** option.
   3. Create the new data source using the SQL below. The SQL produces a list of country/region, state/province names and two character state/province codes.

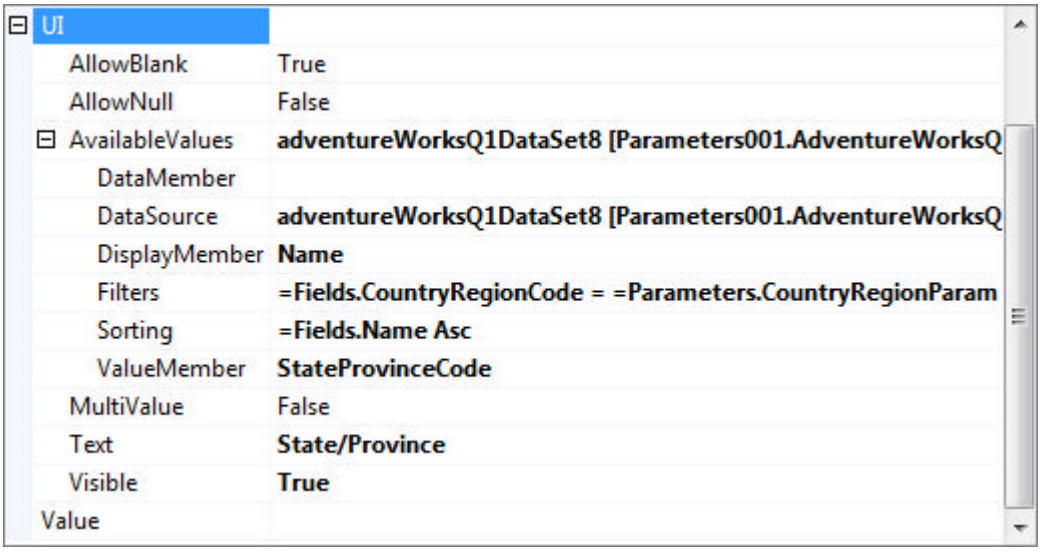
      [SQL] List State/Province

      ```
      SELECT CountryRegionCode, NAME, StateProvinceCode
      FROM Person.StateProvince
      ```

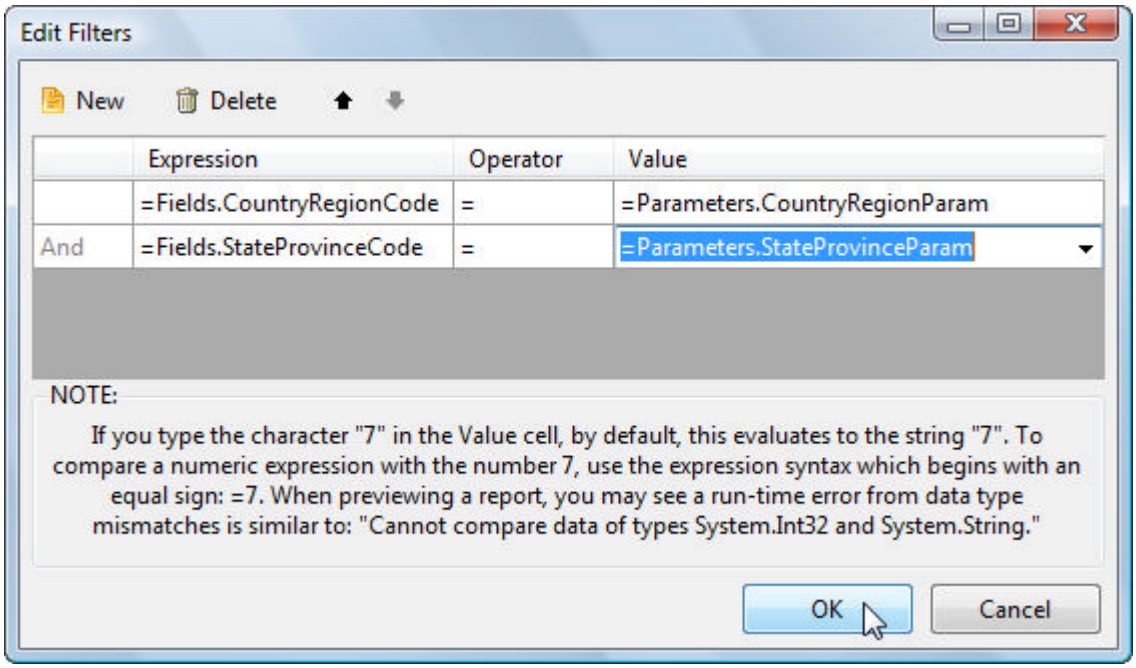
   4. With the DataSource set, drop down the **UI.AvailableValues.DisplayMember** property and select "Name". Drop down the list for **UI.AvailableValues.ValueMember** and select "StateProvinceCode".
   5. Set the **UI.AvailableValues.Sorting** property to "=Fields.Name Asc".
   6. Click the **UI.AvailableValues.Filters** property to display the Edit Filters dialog. Add a new expression where the Fields.CountryRegionCode is equal to the Parameters.CountryRegionParam. This step produces the cascading effect between countries and state/provinces.



The ReportParameter UI properties should look like this screenshot:



5. Close the ReportParam Collection Editor dialog.
6. In the Edit Filters dialog for the report, add two filter expressions as shown in the screenshot below. This step matches the report output to the parameter values:



7. Click the Preview tab and experiment with the automatically created, cascading drop down lists.

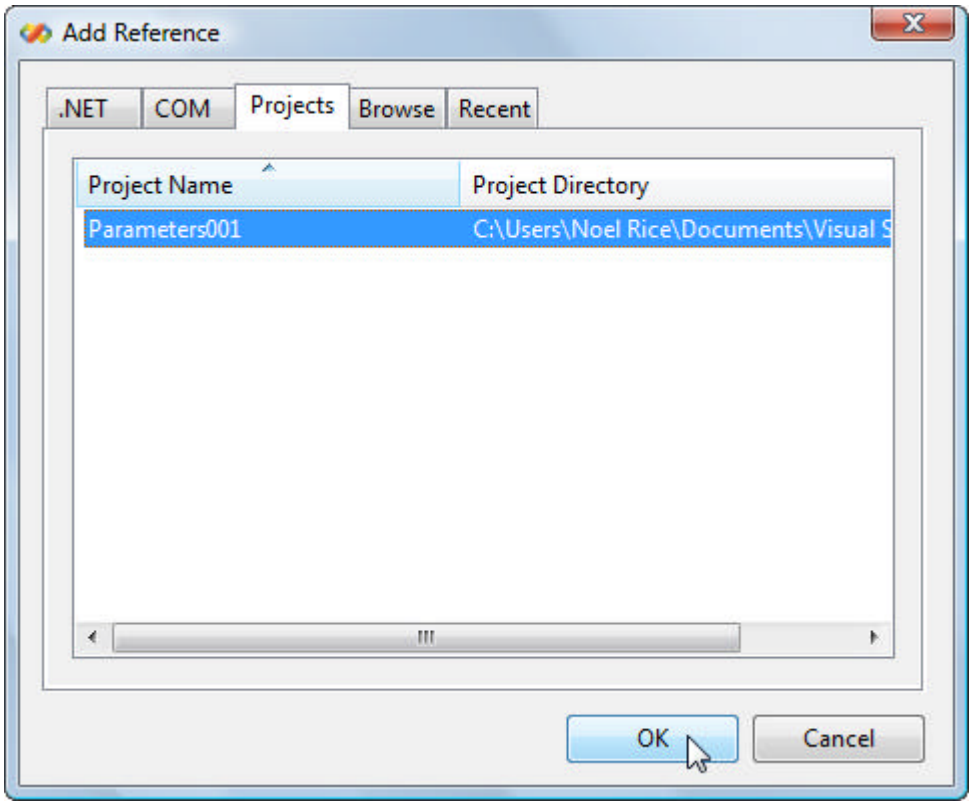
View Parameterized Reports

How much more work is it to run a parameterized report in a separate viewing application? Not very much as it turns out. If you expected to write some code, you'll be disappointed, as all the work happens at design time. The basic steps are:

- Create a windows or web application.
- Add a ReportViewer control to the form or web page.
- Add a reference to the assembly that holds your report assembly.
- Assign the Report property to one of the reports in your assembly. All reports defined in your assembly show up in the Report property drop down list.

Use the class library for the previous "Cascading Parameters" project as a starting point.

1. Add a new Windows Application project to the Visual Studio solution.
2. Add a reference to the "Cascading Parameters" assembly.



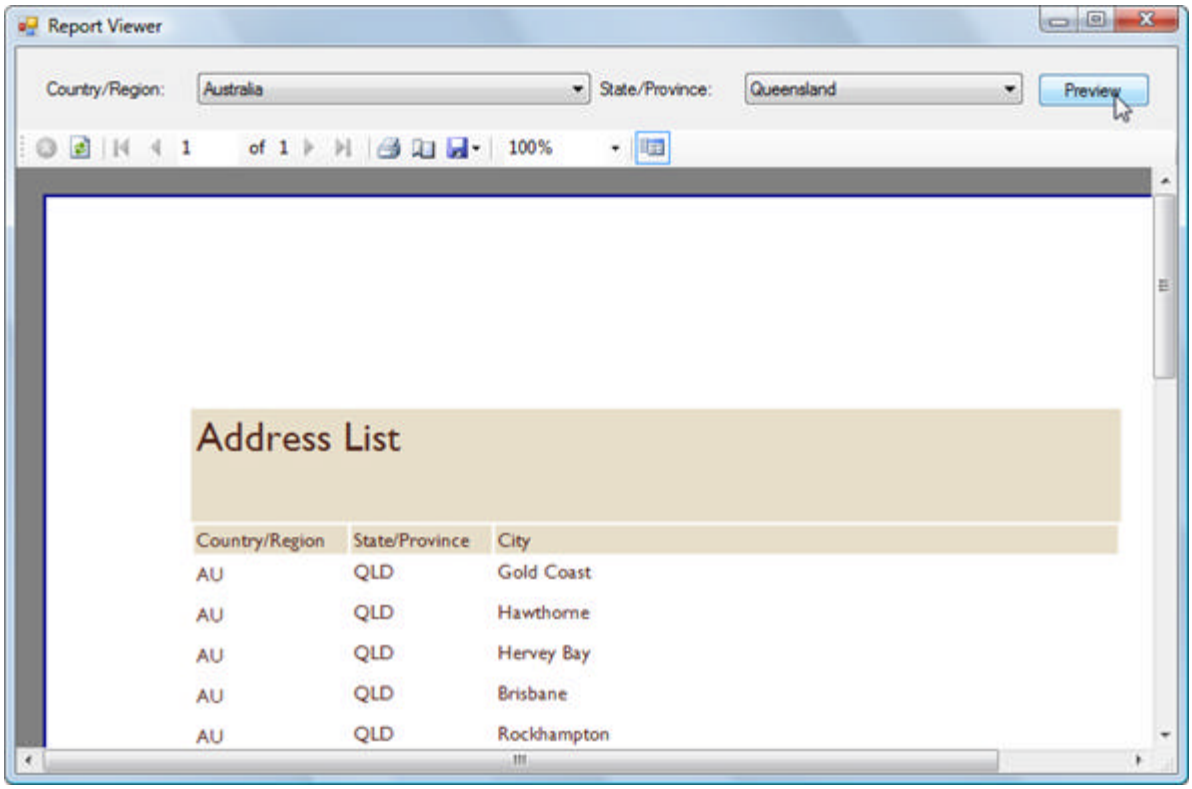
3. Add a ReportViewer control to the default form.
4. In the Properties window, locate the Report property and drop down the list. Select the report that contains the "Cascading Parameters" example.

> When you choose a report, a component representing the report will be created automatically in the component tray. This component allows you to access all the settings of the report including parameters, sorting, grouping and filtering. In this example, you don't need any of this because the UI will be populated automatically without further work on your part.

5. In the Solution Explorer, right-click your viewing application project and select **Set as StartUp Project.**
6. Press **F5** to run the application.
7. In the viewer, click the refresh button ( ).

> If you absolutely must code something, you can handle the refresh in code if you like. In the Form_Load event handler, add:
>
> **[C#] Refreshing the Report**
>
> ```
> private void Form1_Load(object sender, EventArgs e)
> {
>  reportViewer1.RefreshReport();
> }
> ```
>
> **[VB] Refreshing the Report**
>
> ```
> Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
>  reportViewer1.RefreshReport()
> End Sub
> ```

8. Select country/region, then state/province from the drop down lists.
9. Click the **Preview** button.



Summary

In this section you learned how parameters work with expressions to create dynamic, powerful and maintainable reports. You learned to create parameters at design-time and run-time. You also learned to automatically include simple and cascading parameters in your user interface. We created a report viewing winforms application that automatically displays parameter prompts.

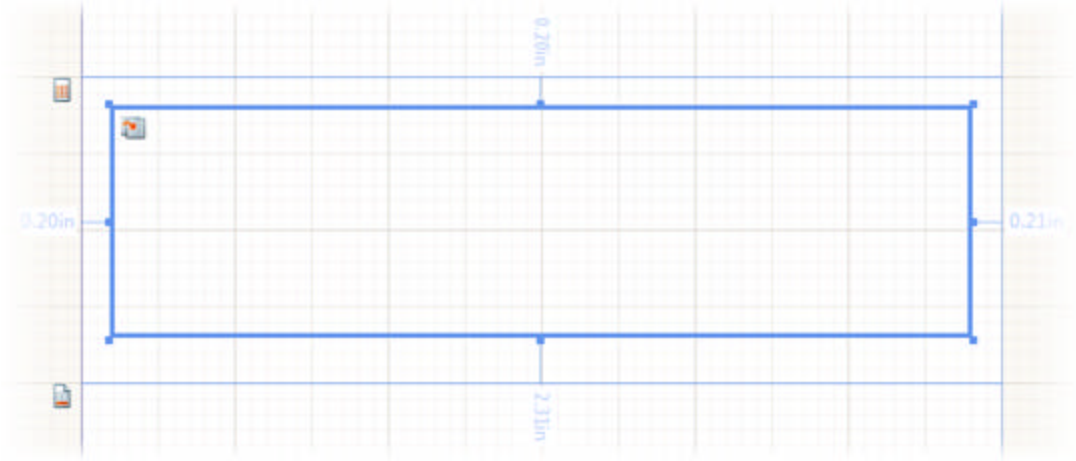
11 Sub Reports

Objectives

This section demonstrates how a SubReport item is used to display one report within another report, allowing you to compose complex reports from multiple report sources. You will learn how to create aggregate reports from disparate report sources, master-detail reports and how to hide SubReports based on expression criteria.

SubReport Basics

The SubReport report item lets you display one report within another report. The data for each SubReport within a master report can be completely different. The screenshot shows a SubReport at design-time.



Some possible uses for SubReport are:

- **Master/Detail** reports: This is a very common use where a SubReport are placed in the detail of the master report and passed parameters to define what data the SubReport should display.
- **Aggregate** reports**:** When you have different batches of data that all need to be displayed in the same report, you can use a SubReport to include each collection of data. For example you could combine a product listing, a sales summary and a list of sales people each in their own sub reports, with each sub report getting data from a different source.

Some of the significant SubReport properties include:

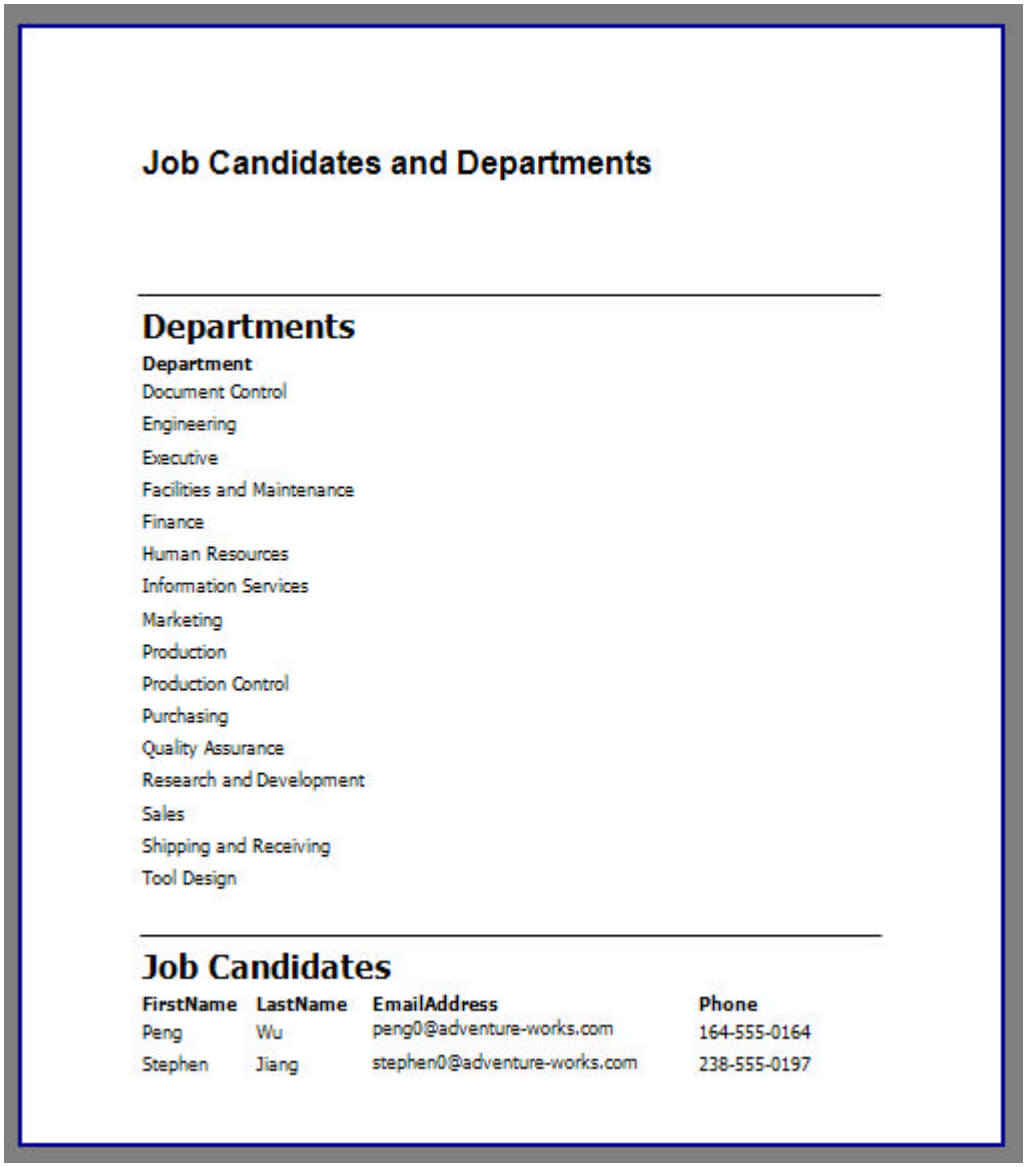
Property	Description
ReportSource	The report to display as a SubReport. Click in this property, and then click the drop down button to select a report from the list.
Visible	Set to **False** to suppress the display of the SubReport.
ConditionalFormatting	A collection containing rules and formatting to be applied to this SubReport if the rules succeed.
Location	Specifies the **X** and **Y** coordinates of the SubReport .
Size	Specifies the **Width** and **Height** properties of the SubReport.
Parameters	This collection is used to specify the ReportParameters property of the report being shown (i.e. the report specified in the **ReportSource** property).

Page Settings:

By design, if a subreport is nested in a report, the subreport page settings are ignored (including paper kind/size, orientation and margins) and only the top-level report page settings are used.

Lab: Aggregate Reports

To create an aggregate of several reports, first create the report definitions that will make up the final product. The report below includes a listing of departments and a second listing of job candidates. The sections of data within this one report have completely different data sources. Using the technique demonstrated in this tutorial, you can combine any number of different reports into a single report.



Job Candidates and Departments

Departments

Department
Document Control
Engineering
Executive
Facilities and Maintenance
Finance
Human Resources
Information Services
Marketing
Production
Production Control
Purchasing
Quality Assurance
Research and Development
Sales
Shipping and Receiving
Tool Design

Job Candidates

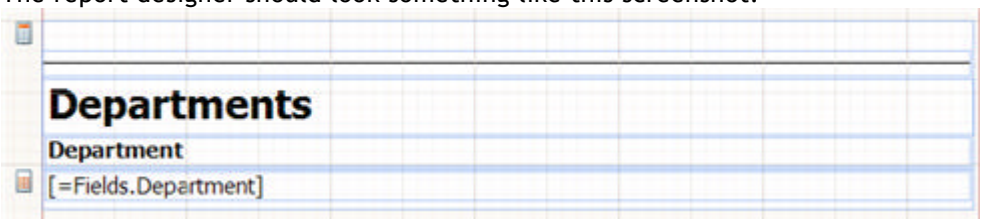
FirstName	LastName	EmailAddress	Phone
Peng	Wu	peng0@adventure-works.com	164-555-0164
Stephen	Jiang	stephen0@adventure-works.com	238-555-0197

1. Create a new Report class and name it JobCandidates.cs.
2. Using the Report Wizard, create a simple listing of departments using the SQL below against the AdventureWorks database.

   [SQL] List Departments

   ```
   SELECT NAME AS Department
   FROM HumanResources.Department
   ```

3. Add a **Shape** report item to the page header. Set the **ShapeType** property to be **Horizontal Line**.
4. Remove the page footer section.
5. The report designer should look something like this screenshot:

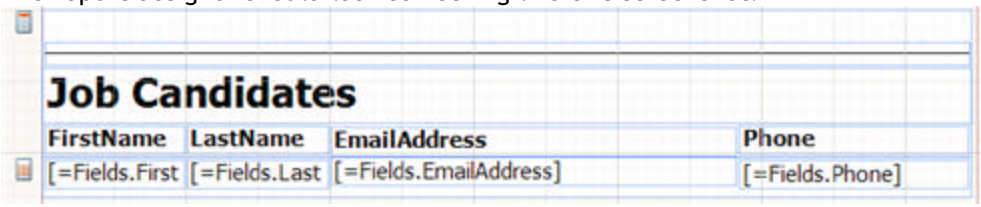


6. Using the Report Wizard, create a list of job candidates using the SQL below against the AdventureWorks database.

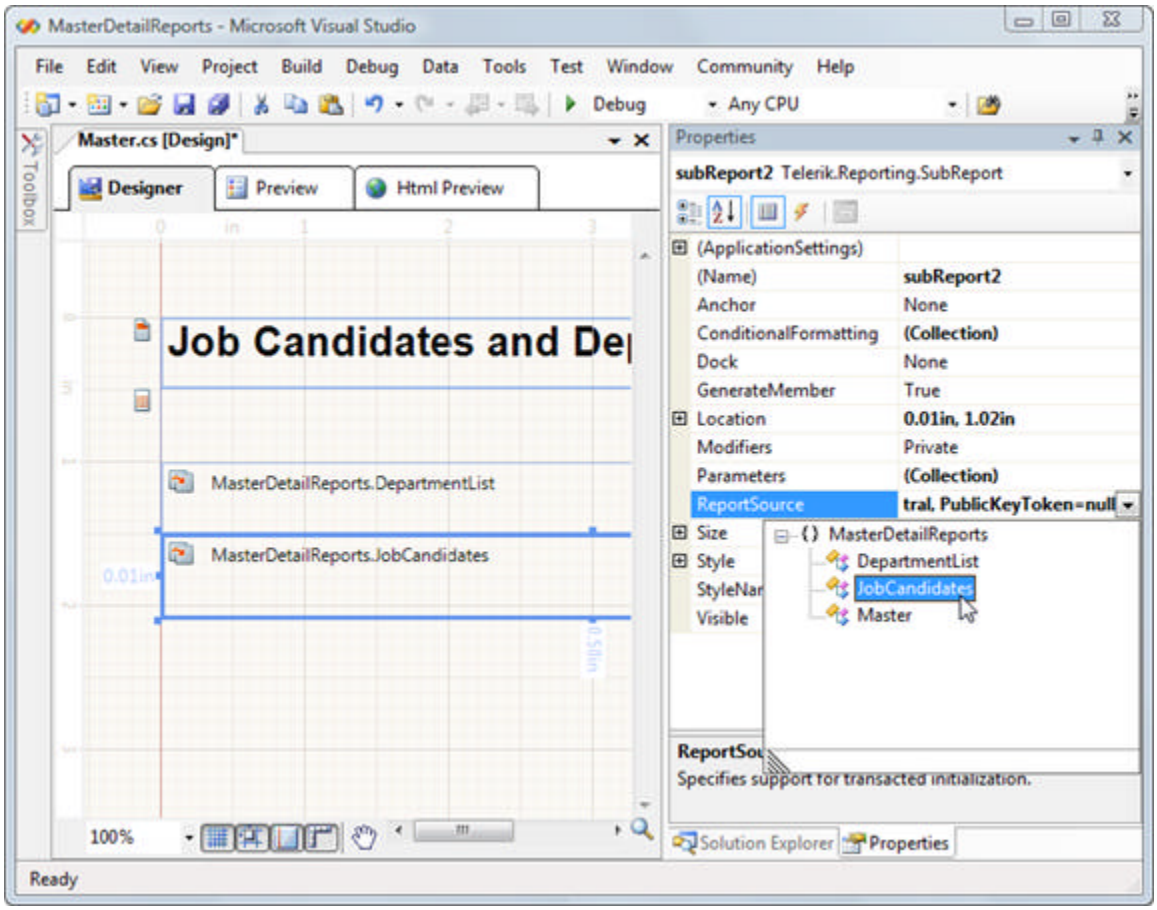
   [SQL] List Departments

```
SELECT
Person.Contact.FirstName,
Person.Contact.LastName,
Person.Contact.EmailAddress,
Person.Contact.Phone
FROM HumanResources.Employee
INNER JOIN Person.Contact
ON HumanResources.Employee.ContactID = Person.Contact.ContactID
INNER JOIN HumanResources.JobCandidate
ON HumanResources.Employee.EmployeeID = HumanResources.JobCandidate.EmployeeID
```

7. Add a **Shape** report item to the page header. Set the **ShapeType** property to be **Horizontal Line**.
8. Remove the page footer section.
9. The report designer should look something like this screenshot:



10. Press **F6** to rebuild the application. This step will make the DepartmentList and JobCandidates classes available to the SubReport items.
11. Create a third report and name it "Master.cs". Cancel the Report Wizard.
12. In the detail section of the report, add two **SubReport** items, one above the other.
13. Select the top SubReport and locate the **ReportSource** property in the Property window. Select the "DepartmentList" report from the drop down list.
14. Select the bottom SubReport and locate the **ReportSource** property in the Property window. Select the "JobCandidates" report from the drop down list.



15. Click the **Preview** tab to view the master report and its sub-reports.

Lab: Master Detail Reports

Master/Detail reports are created using SubReports. The master report passes a parameter to tell the SubReport what detail data it should display. For example, you can list Departments in a master report and all the employees for each department in the detail.

The workflow is very similar to the aggregate report in that you start by defining your detail report, add filtering and parameters to control the filtering. Then you define the master report, add a SubReport, make the SubReport use the detail report as the **ReportSource,** and pass the SubReport the appropriate parameters.

Create the Detail Report

1. Create a new Report class and name it EmployeeDetail.cs.
2. Using the Report Wizard, create a simple listing of employees using the SQL below against the AdventureWorks database.

   [SQL] Employee List

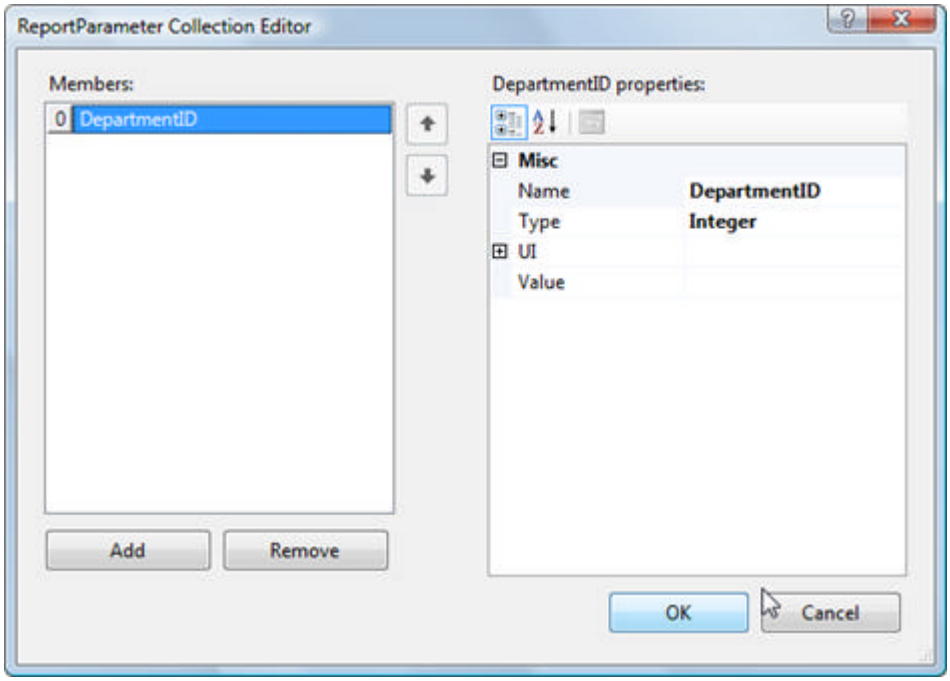
```
SELECT
Person.Contact.FirstName,
Person.Contact.LastName,
HumanResources.EmployeeDepartmentHistory.DepartmentID
FROM HumanResources.Employee
INNER JOIN HumanResources.EmployeeDepartmentHistory
ON HumanResources.Employee.EmployeeID = HumanResources.EmployeeDepartmentHistory.Employee
```

```
INNER JOIN Person.Contact
ON HumanResources.Employee.ContactID = Person.Contact.ContactID
```

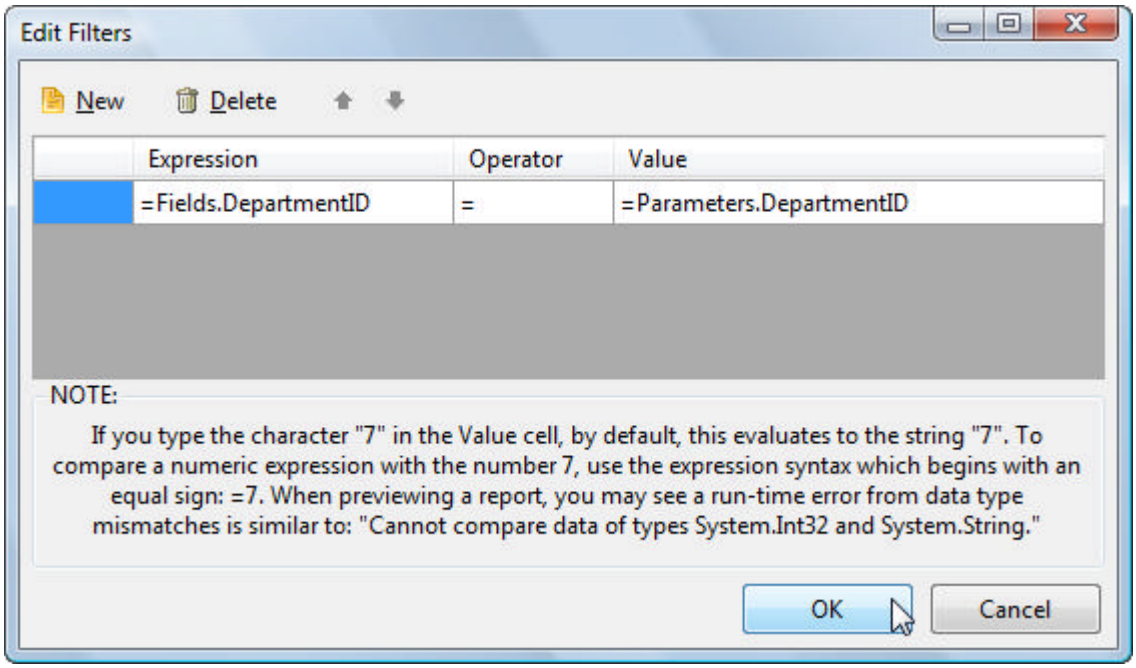
3. Select only the "FirstName" column to display in the report detail.
4. Delete all sections except for the detail.
5. Select the TextBox that represents "FirstName"
6. Double-click the TextBox that represents the FirstName column and edit the expression so that it reads:

   =Trim(Fields.FirstName) + " " + Trim(Fields.LastName)

7. Edit the the **ReportParameters** property: add a new parameter with **Name** "DepartmentID" and **Type** Integer.



8. Edit the **Filters** property so that only records where the DepartmentID is equal to the DepartmentID parameter are displayed.



Create the Master Report

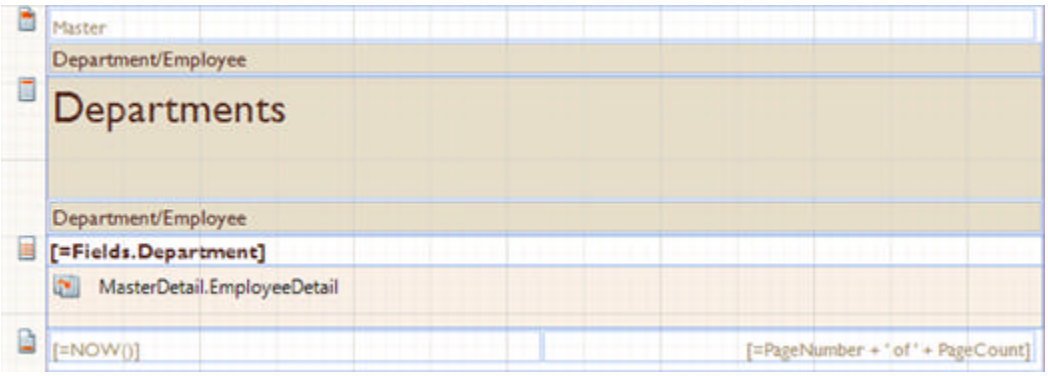
The master report will contain a simple listing of department names followed by a SubReport that contains all the employees, using our EmployeeDetail.cs report.

1. Create a new Report class and name it DepartmentMaster.cs.
2. Using the Report Wizard, create a simple listing of employees using the SQL below against the AdventureWorks database.

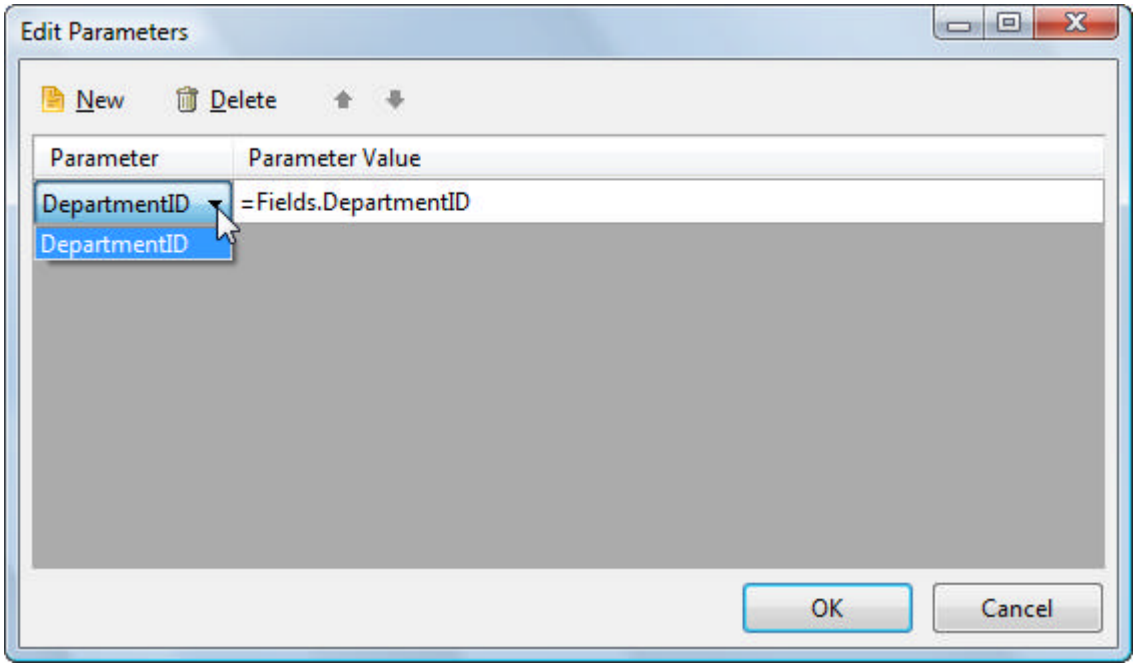
   [SQL] Department List

   ```
   SELECT NAME AS Department, DepartmentID
   FROM HumanResources.Department
   ```

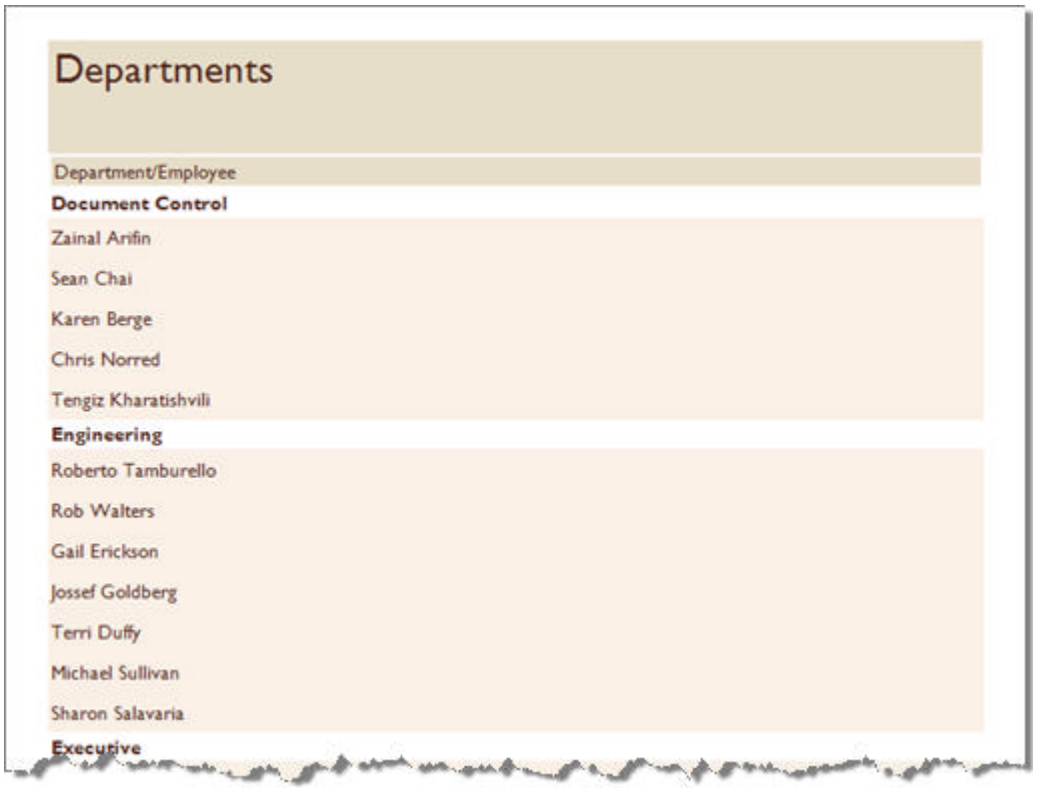
3. Select only the "Department" column to display in the report.
4. In the Report Wizard, use the "Solstice" output style.
5. In the Report Header and Page Header sections you will find a TextBox labeled "Department". Double-Click this TextBox and enter "Department/Employee".
6. Resize the Detail section of the report vertically to make it taller (so there will be room to accept the SubReport).
7. In the detail section, select the "=Fields.Department" TextBox and set the **Dock** property to **Top**.
8. Drop a **SubReport** to the detail section, just below the TextBox for "Department". Set the **Dock** property to **Fill**. Also set the **Style.BackgroundColor** to "Linen" (This will make the detail section stand out slightly).
9. In the designer your master report should look something like this example:



10. Again, select the SubReport and locate the ReportSource property from the Property Window. Select the EmployeeDetail report from the dropdown list.
11. Edit the SubReport **Parameters** property. Add a new parameter where DepartmentID is equal to Fields.DepartmentID. *This step passes the current DepartmentID for the detail row to the SubReport, where the DepartmentID is used for filtering.*



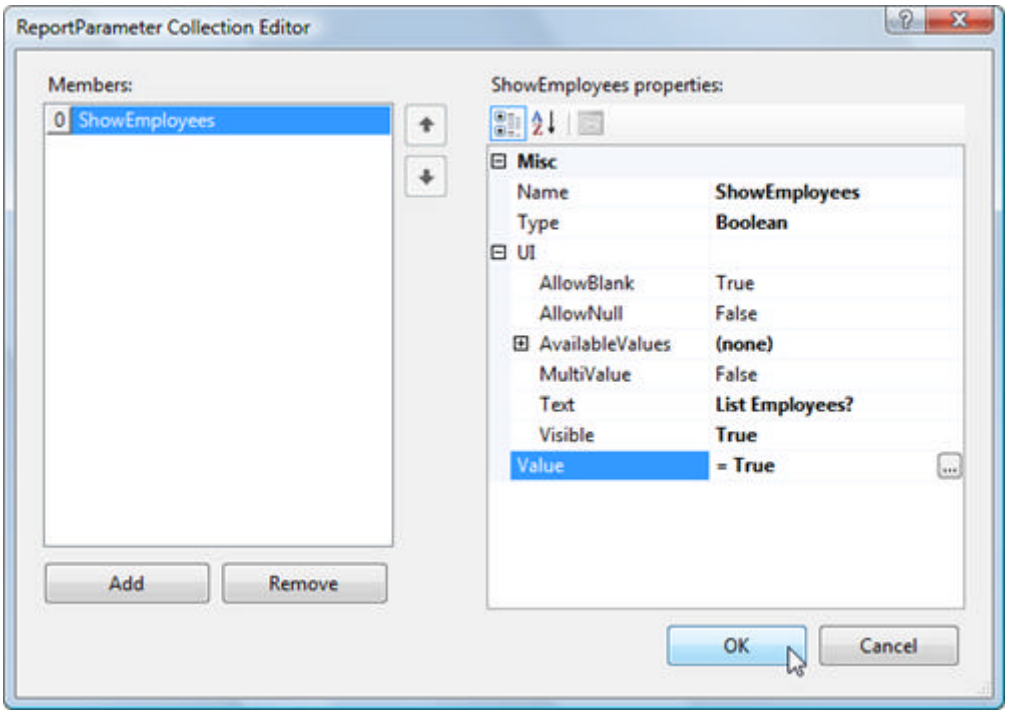
12. Click the **Preview** tab to see the master/detail report.



Lab: Hiding SubReports

Use the SubReport **Visible** property to dynamically hide or show a SubReport at runtime. You can use report parameters to signal which SubReports should be displayed. You can modify the SubReports "Master Detail Reports" example to optionally display employees.

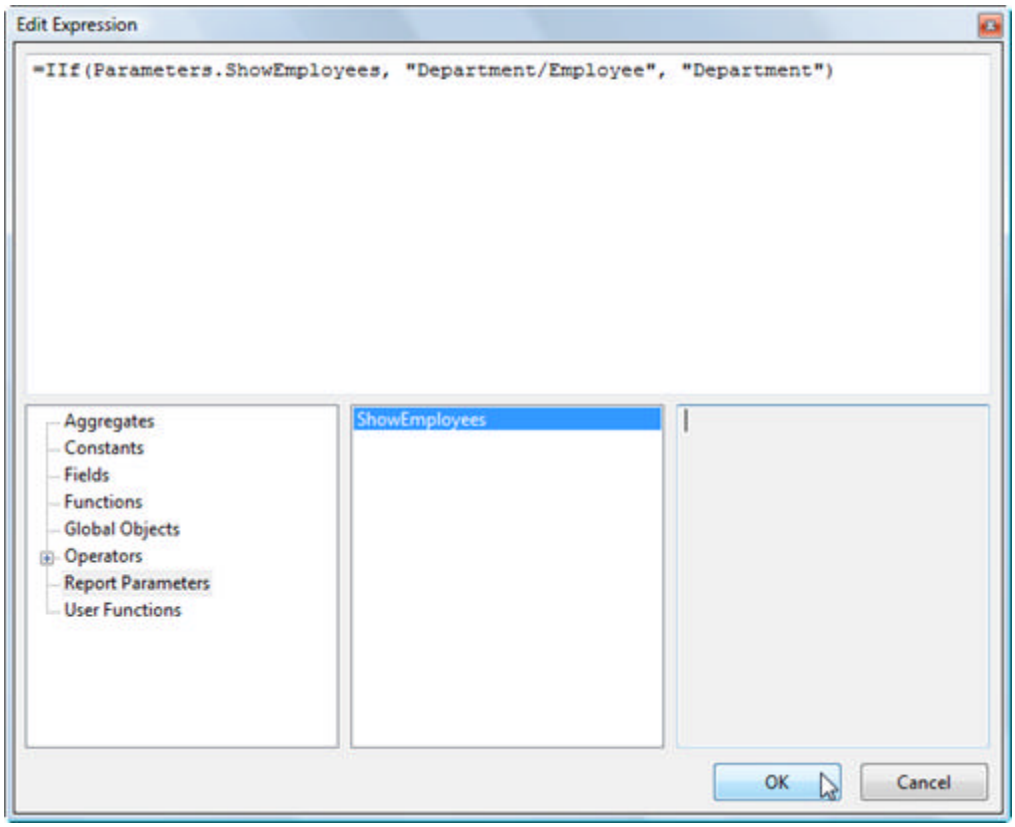
1. Add a parameter to the master report that controls if they employees are shown or not.
    1. Open the editor for the master report ReportParameters property.
    2. Add a parameter and set the **Name** property to "ShowEmployees".
    3. Set the **Type** property to "Boolean"
    4. Set the **UI.Visible** property to **True**
    5. Set the **Value** property to "= True".



2. Select the TextBox in the ReportHeader that is labeled "Departments". Set the Value property to this expression:

   =IIf(Parameters.ShowEmployees, "Department/Employee", "Department")

   You can enter it directly or use the Edit Expression dialog:



3. In the master report, select the SubReport item. In the Properties Window, go to the Events tab and double-click the ItemDataBound event to create a handler. Replace the handler with the code below that sets the SubReport (the Telerik.Reporting.Processing version of SubReport) to the "ShowEmployees" value.

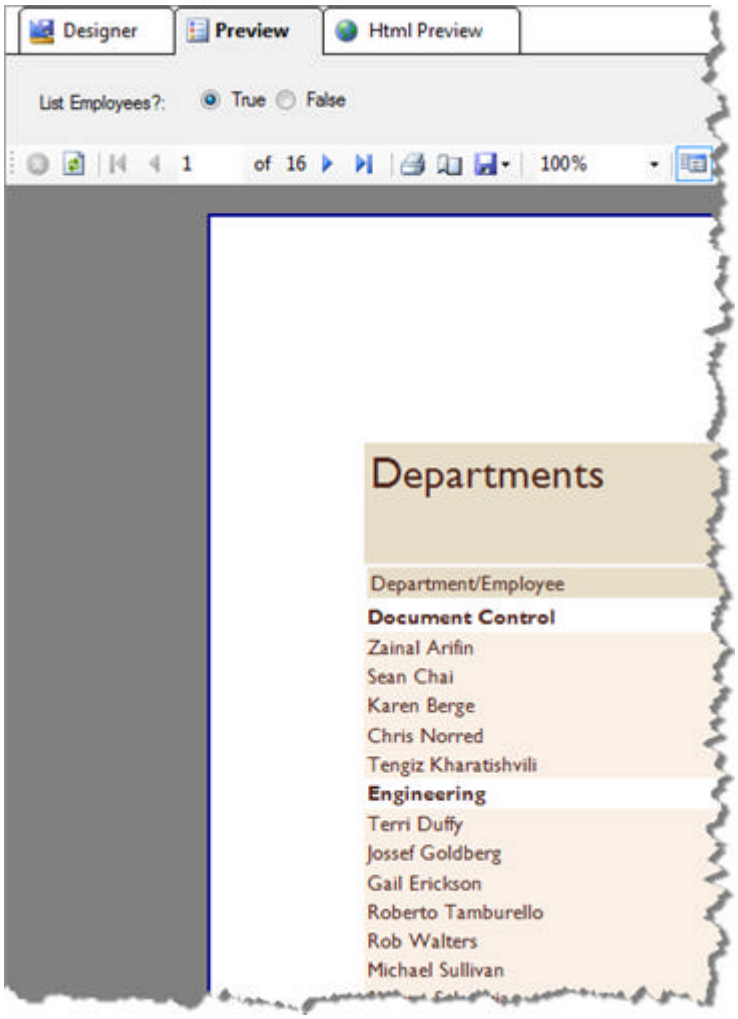
[C#] Setting the SubReport Visible Property

```
private void subReport1_ItemDataBound(object sender, System.EventArgs e)
{
   (sender as Telerik.Reporting.Processing.SubReport).Visible =
       (bool)this.ReportParameters["ShowEmployees"].Value;
}
```

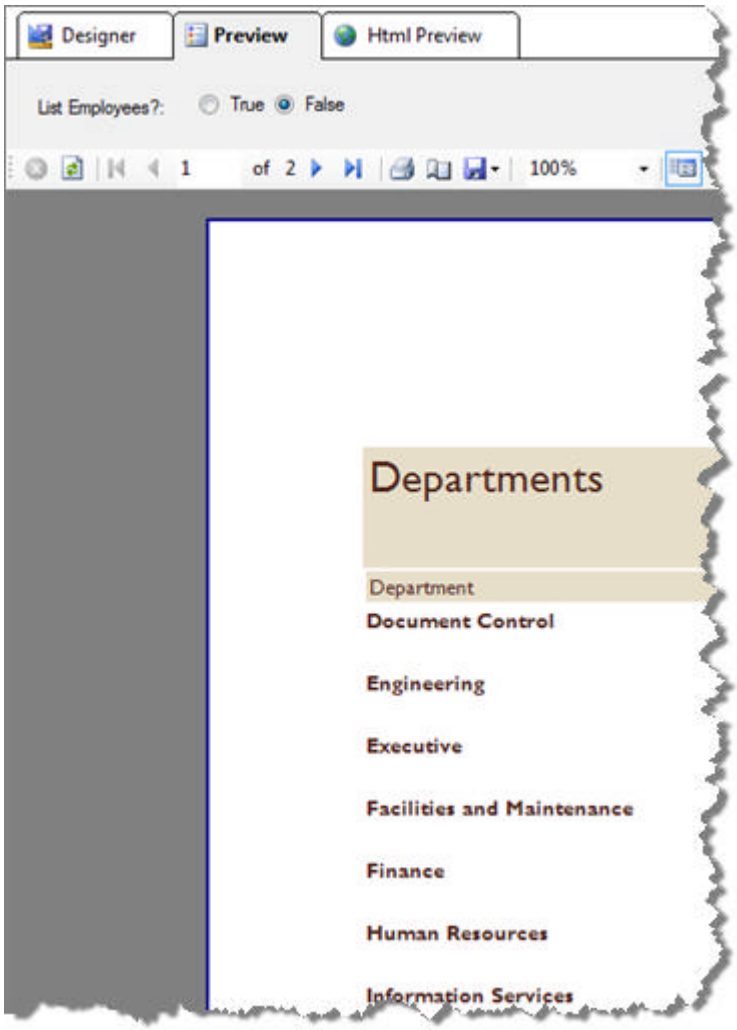
[VB] Setting the SubReport Visible Property

```
Private Sub subReport1_ItemDataBound(ByVal sender As Object, ByVal e As System.EventArgs)
 (TryCast(sender, Telerik.Reporting.Processing.SubReport)).Visible = DirectCast
(Me.ReportParameters("ShowEmployees").Value, Boolean)
End Sub
```

4. Click the **Preview** button to display the report. With the "ShowEmployee" set to True, the report shows master and detail records (and the column heading shows "Department/Employee"):



...with "ShowEmployees" set to False the report shows departments only. Notice the column heading displays "Department".



Summary

This section demonstrated how SubReports are used in situations where Master-detail, aggregate reports and where ever report data can't be included in a single report. We also looked at how to conditionally hide or show SubReports based on the value of an expression.

12 Styling Your Report

Objectives

This section explores the array of choices in Telerik Reporting for altering the appearance of your report using styles and style rules. You will learn how to define a style sheet, to export and to re-use the stylesheet in other reports.

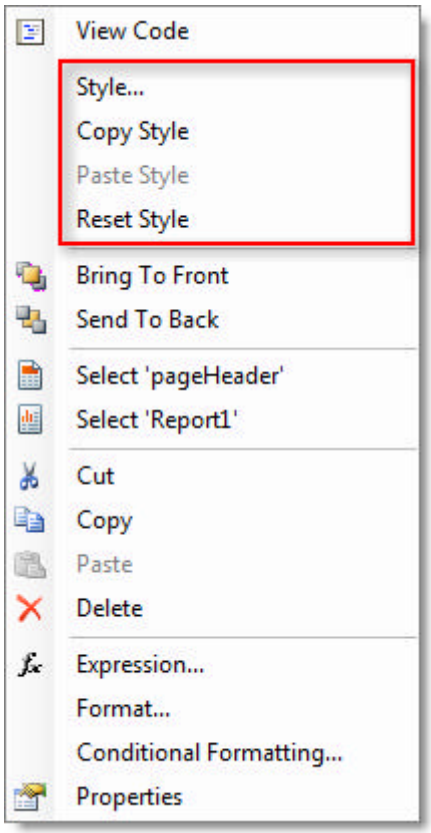
Styling Basics

Telerik Reporting uses a built-in styling model that is similar to CSS. The model provides for very fine-grained visual customization of all elements of a report directly in the Visual Studio designer. This CSS-like mechanism offers full control over such things as the background, colors, borders, and images for every item on your report. You can assign styles by using CSS selectors that point to report items based on their type, the attributes of the item, a style name, or if the report item is a descendant of a given report item.

Like CSS, styles can be defined globally, inherited from parent items and can also be explicitly defined for a report item. A child report item placed within another report item derives its styling based on the following precedence:

- **Lowest Priority - Parent:** If the parent control has a style applied, the child inherits the parent's styling.
- **Second Priority - Global:** If any of the global styles apply to the child then the control will use this styling. Parent styling is ignored in this case.
- **Highest Priority - Inline:** If the child item's style properties have changed from their defaults, the child style properties are honored. Global styling and parent style are ignored in this case.

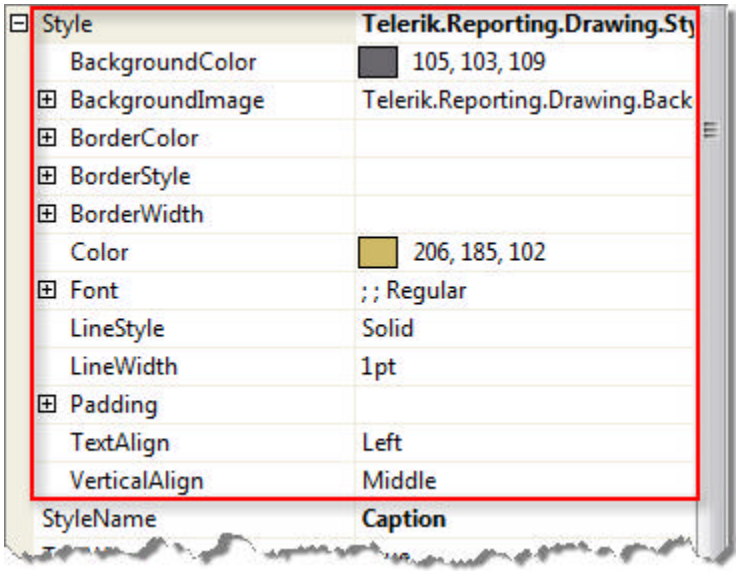
At design time you can use the styling mechanism without needing to understand much of the underlying mechanics. Right-click a report item to get the style related context menu options:



From the context menu you can select:

- **Style...**: Displays the **Edit Style** dialog so that you can change the **Text**, **BackGround**, **Edges** and **Line** styles.
- **Copy Style**: Persists all the style information for a report item.
- **Paste Style**: Applies all the styling from one report item to another. This becomes enabled once the Copy Style menu option has been selected once. You can copy and paste styles between disparate items, e.g. copy the style from a TextBox and paste it to a section.
- **Reset Style**: Restores the style properties to the global style.

Or, you can also use the **Style** property of any report item to invoke the Edit Style dialog (using the ellipses), or edit each of the sub properties individually:

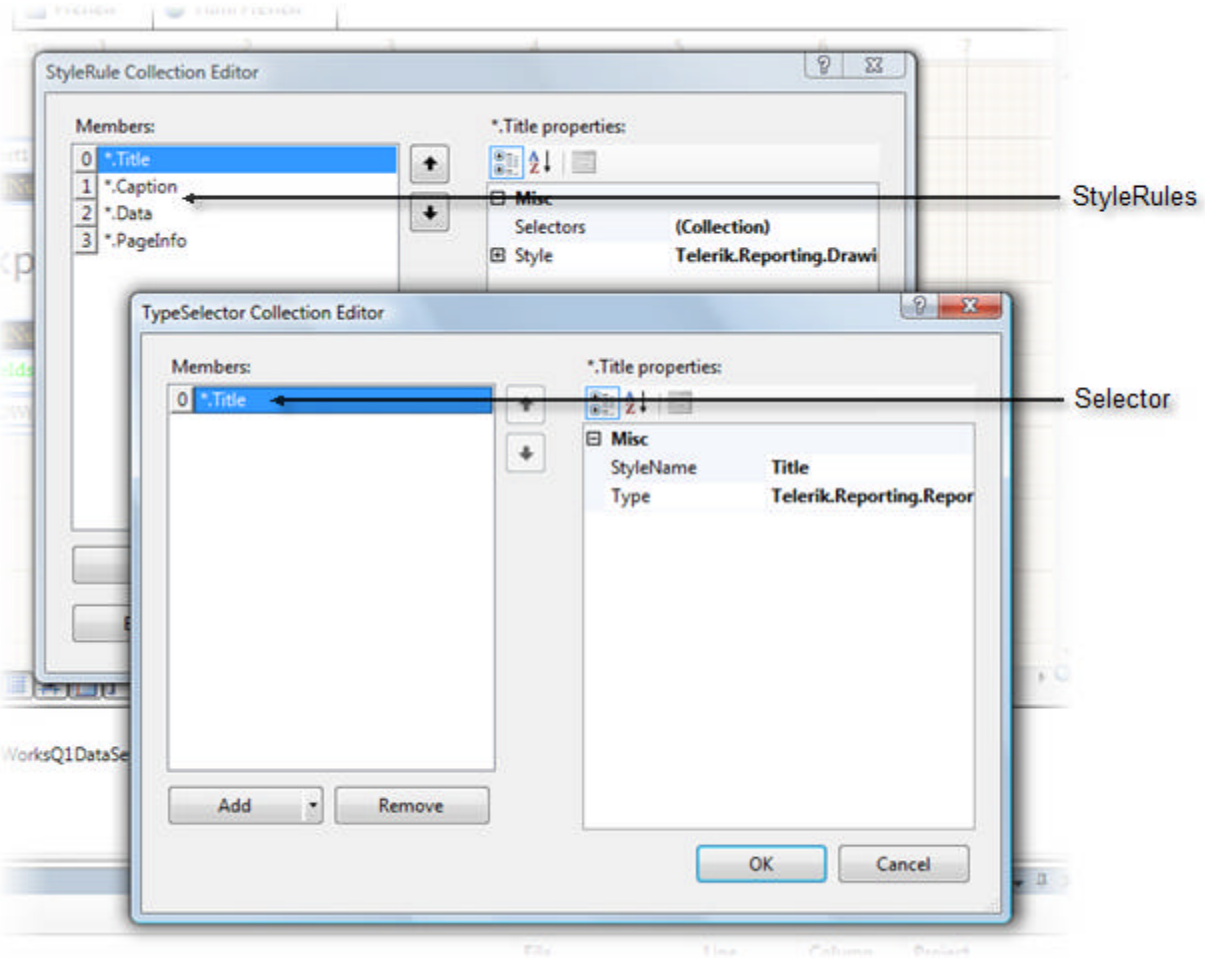


Style Rules

The styling mechanism goes deeper than the simple direct assign of style properties. Telerik Reporting offers a much more flexible, CSS-like, capability than the simple choices available in the **Style** property alone. Telerik Reporting can flexibly handle entire categories of report items by using **Style Rules** that determine what items a style is applied to. A **Style Rule** is defined using "Selectors" (i.e. criteria for applying a style):

1. **TypeSelectors** target all report items of a particular type (for example **TextBox** or **DetailSection**).
2. **StyleSelectors** are named styles that can be applied to report items through the report item's **StyleName** property.
3. **AttributeSelectors** target all report items with a particular attribute (for example **Font Family= Verdana**).
4. **DescendantSelectors** target report items which descend from (or, are placed within) other report items. For example you can define a style that should be applied to all TextBoxes in the report's **DetailSection**.

Use the Report object's **StyleSheet** property to invoke the **StyleRule Collection Editor**. When you create a report using the Report Wizard, the wizard will pre-define several style rules. The screen shot below shows style rules defined for a simple listing report in the "Apex" style. Each StyleRule has a collection of **Selectors** and a **Style** property. The Selectors determine when the style should apply and the Style property defines the appearance of the selected item. In this example, all four rules have Selectors that use a named style as a criteria.



Lab: Adding Style Rules

Imagine taking the "ProductReviews" table from the AdventureWorks database and just dumping it to the page. Looks pretty dry, doesn't it? Not much going on there, nothing draws your eye, the information runs together and the overall effect is not very readable. Also notice that the "Comments" column seems to have extra line feeds that heighten the raggedy feel.

Product Reviews

Product Name	Rating	Comments
Mountain Bike Socks, M	5	I can't believe I'm singing the praises of a pair of socks, but I just came back from a grueling 3-day ride and these socks really helped make the trip a blast. They're lightweight yet really cushioned my feet all day. The reinforced toe is nearly bullet-proof and I didn't experience any problems with rubbing or blisters like I have with other brands. I know it sounds silly, but it's always the little stuff (like comfortable feet) that makes or breaks a long trip. I won't go on another trip without them!
HL Mountain Pedal	4	A little on the heavy side, but overall the entry/exit is easy in all conditions. I've used these pedals for more than 3 years and I've never had a problem. Cleanup is easy. Mud and sand don't get trapped. I would like them even better if there was a weight reduction. Maybe in the next design. Still, I would recommend them to a friend.
HL Mountain Pedal	2	Maybe it's just because I'm new to mountain biking, but I had a terrible time getting use to these pedals. In my first outing, I wiped out trying to release my foot. Any suggestions on

We can fix most these problems with the judicious use of styles and Style Rules. The key word here is "judicious" - if you go crazy with styles, you can still end up with with a visual mish-mash. A little restraint goes a long way. Also, keep in mind what you're trying to accomplish here:

- Provide visual cues to the reader as to where the title, column headers and content are within the page. They shouldn't have to think about it.
- Make the report readable. Background and text colors should contrast so the reader doesn't go blind.
- Of course, to make the report attractive. Here again, not overdoing loud or contrasting colors, heavy or ornate font styles and shapes for the sake of shapes, will make the report more usable on a day-to-day basis. Remember how web sites looked "back in the day", with large red fonts, broad beveled borders and the liberal use of blinking banners? Avoid that pattern and keep the report looking professional.

Creating the Basic Report

These steps create the report with minimal Style formatting for specific items.

1. Create a new report, but cancel the Report Wizard.
2. Select the report in the designer and in the Properties Window open the DataSource Property drop down list. Select the Add Project Datasource option.
3. Configure the datasource to use the SQL below against the AdventureWorks database:

   [SQL] List Product Reviews

   ```
   SELECT
    Production.Product.NAME,
    Production.ProductReview.Rating,
    Production.ProductReview.Comments
   FROM
    Production.ProductReview
   INNER JOIN Production.Product
   ON Production.ProductReview.ProductID = Production.Product.ProductID
   ```

4. Set the **SnapGridSize** property of the report to **.05in.** This will make it easier to move report items around in smaller increments. Set the report Style.Font.Size property to **9pt.** Note: This sets the default size for all fonts in the report.
5. Set the **Height** property for each section where PageHeaderSection = **.7**, DetailSection = **.3** and

PageFooterSection = **.3**.

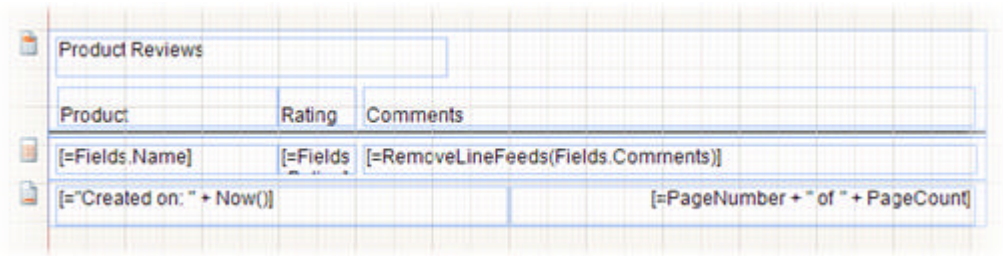
6. In the detail section add bound TextBox items for "Name", "Rating" and "Comments". Use the Data Explorer from the Telerik Reporting menu to drag and drop the bound fields.
   Change the **Value** property for "Comments" to the following expression (a user defined function "RemoveLineFeeds()" will be defined later):

   =RemoveLineFeeds(Fields.Comments)
7. Select the detail section and set the **Style.Padding** property to .05 for **Top** and .03 for **Bottom**. Note: This step provides a little space between blocks of text and keeps them from running together.
8. In the header section, drop a **Shape** report item, set the **ShapeType** to **Horizontal Line,** set the **Dock** property to **Bottom** and the **Size.Height** property to **.05in.** Note: This will provide an underline for the column headings and will help visually separate the heading from the report detail.
9. In the header, above the horizontal line, place three **TextBox** items, lined up above the bound fields in the detail section. Change the text for each TextBox to "Product", "Rating" and "Comments", respectively.
10. Add a **TextBox** to the top of the the PageHeaderSection. Set the **Value** property to "Product Reviews".
11. In the PageFooterSection add two TextBoxes, roughly splitting the horizontal available space between them. Set the Value of the left-most TextBox to:

    ="Created on: " + Now()

    ...And set the rightmost TextBox Value to:

    =PageNumber + " of " + PageCount
12. Also in the rightmost PageFooterSection TextBox, set the **Style.TextAlign** property to **Right**.
13. The page layout should look something like this:



Add a User Defined Function

Styles will solve most of our problems, but not the problem of extra line feeds in the data. For that, a user defined function will serve nicely. Notice the TextBox in the Comments column with a value that reads **RemoveLineFeeds()**? In this step we define that function.

1. Navigate to the code-behind for the report (right-click the designer and select **View Code** from the context menu)
2. Add the following public, static function to your report class:

   [C#] Adding the User-Defined Function

```
public static string RemoveLineFeeds(string text)
{
   string result = text.Replace("\r\n", " ");
   result = result.Replace("\r", " ");
   result = result.Replace("\n", " ");
   return result;
```

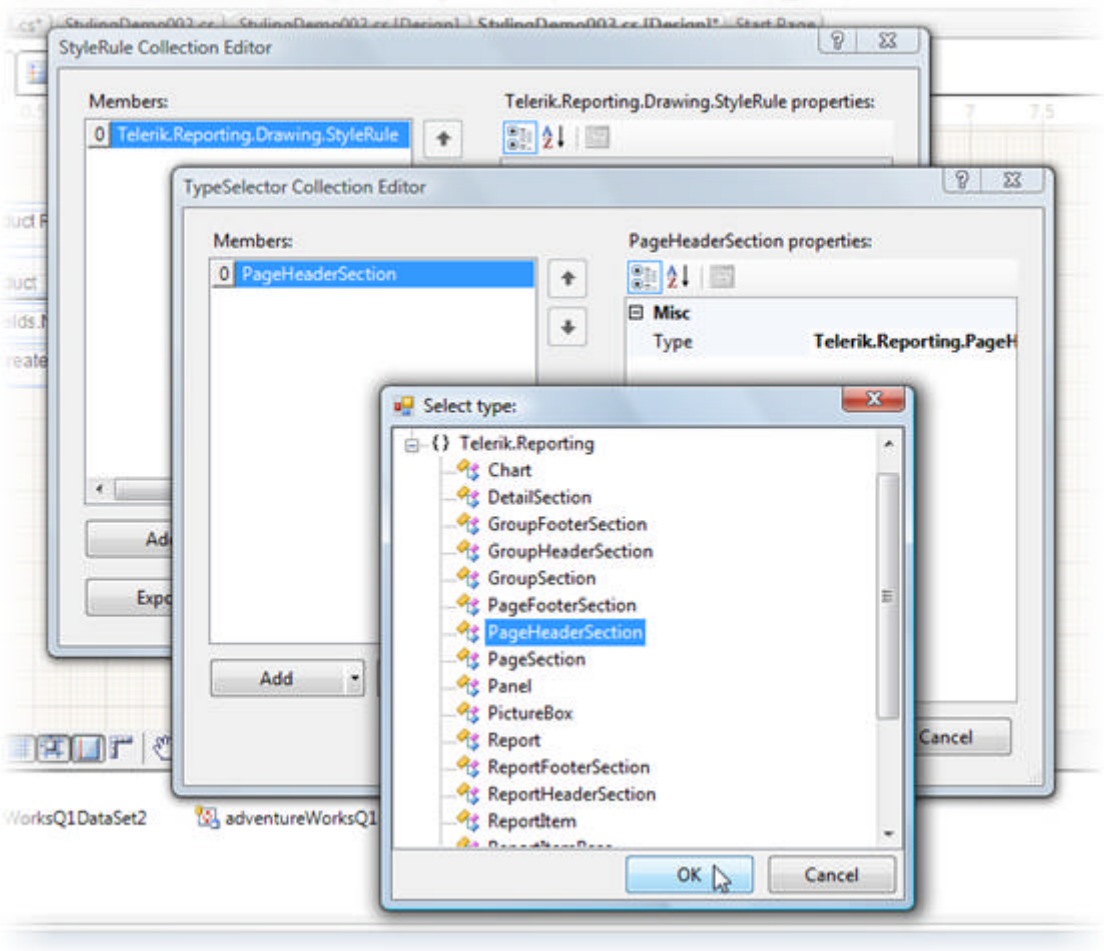
```
}
```

[VB] Adding the User-Defined Function

```
Public Shared Function RemoveLineFeeds(ByVal text As String) As String
 Dim result As String = text.Replace("" & Chr(13) & "" & Chr(10) & "", " ")
 result = result.Replace("" & Chr(13) & "", " ")
 result = result.Replace("" & Chr(10) & "", " ")
 Return result
End Function
```

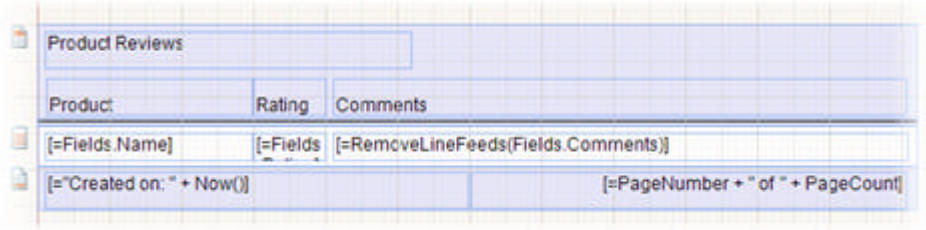
Creating and Applying the StyleSheet

1. In the designer, select the report object. You can do this by clicking outside the report sections in the designer or clicking the box located in the upper left of the report designer.
2. In the Properties Window, click the ellipses of the **StyleSheet** property. This will display the StyleRule collection editor.
3. Create a style rule for the page header section:
    1. Click the **Add** button to create a StyleRule.
    2. Click the **Selectors** property ellipses. This will display the TypeSelector Collection Editor dialog.
    3. Click the **Add** button drop down arrow and choose **TypeSelector.**
    4. Click the **Type** property ellipses. This will display the **Select Type** dialog.
    5. Select the **PageHeaderSection** type and click **OK** to close the Select Type dialog.

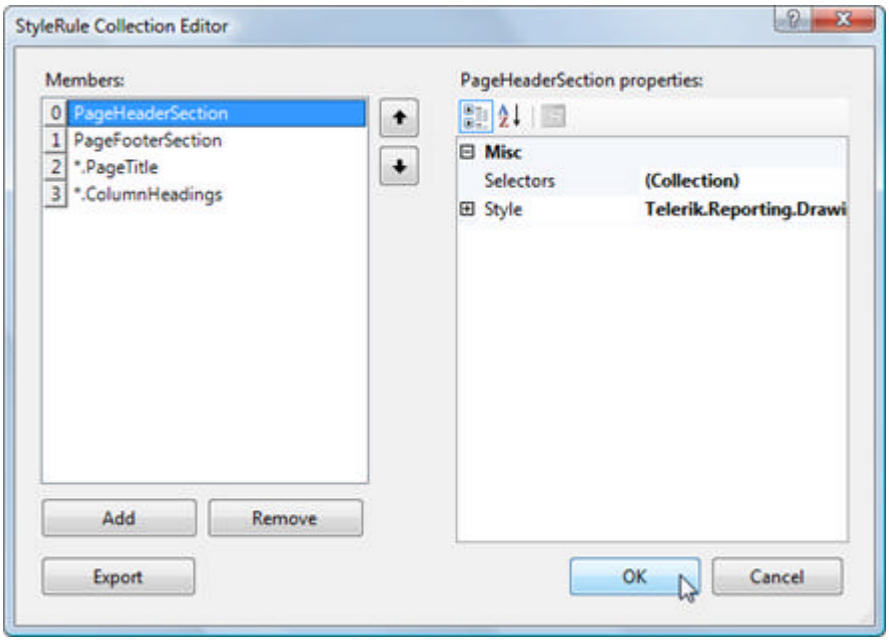


6. Click **OK** to close the TypeSelector Collection Editor dialog.
7. In the StyleRule Collection Editor, open the **Style** property. Select the **BackColor** sub-property and choose "Lavender" from the list of **Web** colors.

4. Create a style rule for the page footer section:
   1. Click the **Add** button to create a StyleRule.
   2. Click the **Selectors** property ellipses. This will display the TypeSelector Collection Editor dialog.
   3. Click the **Add** button drop down arrow and choose **TypeSelector.**
   4. Click the **Type** property ellipses. This will display the **Select Type** dialog.
   5. Select the **PageFooterSection** type and click **OK** to close the Select Type dialog.
   6. Click **OK** to close the TypeSelector Collection Editor dialog.
   7. In the StyleRule Collection Editor, open the **Style** property. Select the **BackColor** sub-property and choose "Lavender" from the list of **Web** colors.
   8. Click **OK** to close the StyleRule Collection Editor and see your changes in the designer.

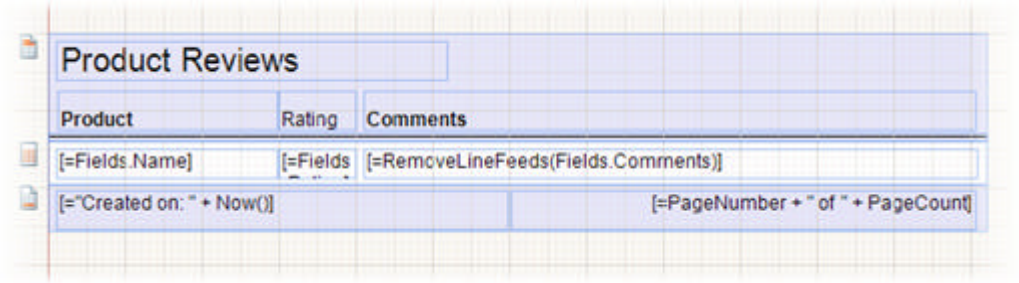


5. Create a style rule for the page title:
    1. In the Properties Window, click the ellipses of the **StyleSheet** property to display the StyleRule collection editor.
    2. Again, use the report **StyleSheet** property to open the StyleRuleCollection Editor.
    3. Click the **Add** button to create a StyleRule.
    4. Click the **Selectors** property ellipses. This will display the TypeSelector Collection Editor dialog.
    5. Click the **Add** button drop down arrow and choose **StyleSelector.**
    6. Set the **StyleName** property to "PageTitle".
    7. Click **OK** to close the TypeSelector Collection Editor dialog.
    8. In the StyleRule Collection Editor, open the **Style** property. Set the **Font.Size** sub-property to **15pt**.
    9. Click **OK** to close the StyleRule Collection Editor. Your changes will not be visible yet.
    10. Select the **TextBox** item from the upper left corner (it should read "Product Reviews").
    11. Set the **StyleName** property to "PageTitle". Your changes should take effect.
6. Create a style rule for the column headings:
    1. In the Properties Window, click the ellipses of the **StyleSheet** property to display the StyleRule collection editor.
    2. Again, use the report **StyleSheet** property to open the StyleRuleCollection Editor.
    3. Click the **Add** button to create a StyleRule.
    4. Click the **Selectors** property ellipses. This will display the TypeSelector Collection Editor dialog.
    5. Click the **Add** button drop down arrow and choose **StyleSelector.**
    6. Set the **StyleName** property to "ColumnHeadings".
    7. Click **OK** to close the TypeSelector Collection Editor dialog.
    8. In the StyleRule Collection Editor, open the **Style** property. Set the **Font.Bold** sub-property to **true**. The StyleRule Collection Editor should at this point look like the screenshot below:



9. Click **OK** to close the StyleRule Collection Editor. Your changes will not be visible yet.
10. Select all three column heading **TextBox** items (hint: hold down the Control key and click each of the TextBox items with the mouse).
11. Set the **StyleName** property to "ColumnHeadings". Your changes should take effect.

7. The designer should look something like the example below:



8. Click the **Preview** tab to see the report:

Product Reviews

Product	Rating	Comments
Mountain Bike Socks, M	5	I can't believe I'm singing the praises of a pair of socks, but I just came back from a grueling 3-day ride and these socks really helped make the trip a blast. They're lightweight yet really cushioned my feet all day. The reinforced toe is nearly bullet-proof and I didn't experience any problems with rubbing or blisters like I have with other brands. I know it sounds silly, but it's always the little stuff (like comfortable feet) that makes or breaks a long trip. I won't go on another trip without them!
HL Mountain Pedal	4	A little on the heavy side, but overall the entry/exit is easy in all conditions. I've used these pedals for more than 3 years and I've never had a problem. Cleanup is easy. Mud and sand don't get trapped. I would like them even better if there was a weight reduction. Maybe in the next design. Still, I would recommend them to a friend.
HL Mountain Pedal	2	Maybe it's just because I'm new to mountain biking, but I had a terrible time getting use to these pedals. In my first outing, I wiped out trying to release my foot. Any suggestions on ways I can adjust the pedals, or is it just a learning curve thing?
Road-550-W Yellow, 40	5	The Road-550-W from Adventure Works Cycles is everything it's advertised to be. Finally, a quality bike that is actually built for a woman and provides control and comfort in one neat package. The top tube is shorter, the suspension is weight-tuned and there's a much shorter reach to the brake levers. All this adds up to a great mountain bike that is sure to accommodate any woman's anatomy. In addition to getting the size right, the saddle is incredibly comfortable. Attention to detail is apparent in every aspect from the frame finish to the careful design of each component. Each component is a solid performer without any fluff. The designers clearly did their homework and thought about size, weight, and funtionality throughout. And at less than 19 pounds, the bike is manageable for even the most petite cyclist. We had 5 riders take the bike out for a spin and really put it to the test. The results were consistent and very positive. Our testers loved the manuverability and control they had with the redesigned frame on the 550-W. A definite improvement over the 2002 design. Four out of five testers listed quick handling and responsivness were the key elements they noticed. Technical climbing and on the flats, the bike just cruises through the rough. Tight corners and obstacles were handled effortlessly. The fifth tester was more impressed with the smooth ride. The heavy-duty shocks absorbed even the worst bumps and provided a soft ride on all but the nastiest trails and biggest drops. The shifting was rated superb and typical of what we've come to expect from Adventure Works Cycles. On descents, the bike handled flawlessly and tracked very well. The bike is well balanced front-to-rear and frame flex was minimal. In particular, the testers noted that the brake system had a unique combination of power and modulation. While some brake setups can be overly touchy, these brakes had a good amount of power, but also a good feel that allows you to apply as little or as much braking power as is needed. Second is their short break-in period. We found that they tend to break-in well before the end of the first ride; while others take two to three rides (or more) to come to full power. On the negative side, the pedals were not quite up to our tester's standards. Just for fun, we experimented with routine maintenance tasks. Overall we found most operations to be straight forward and easy to complete.

Created on: 5/6/2008 2:24:03 PM 1 of 2

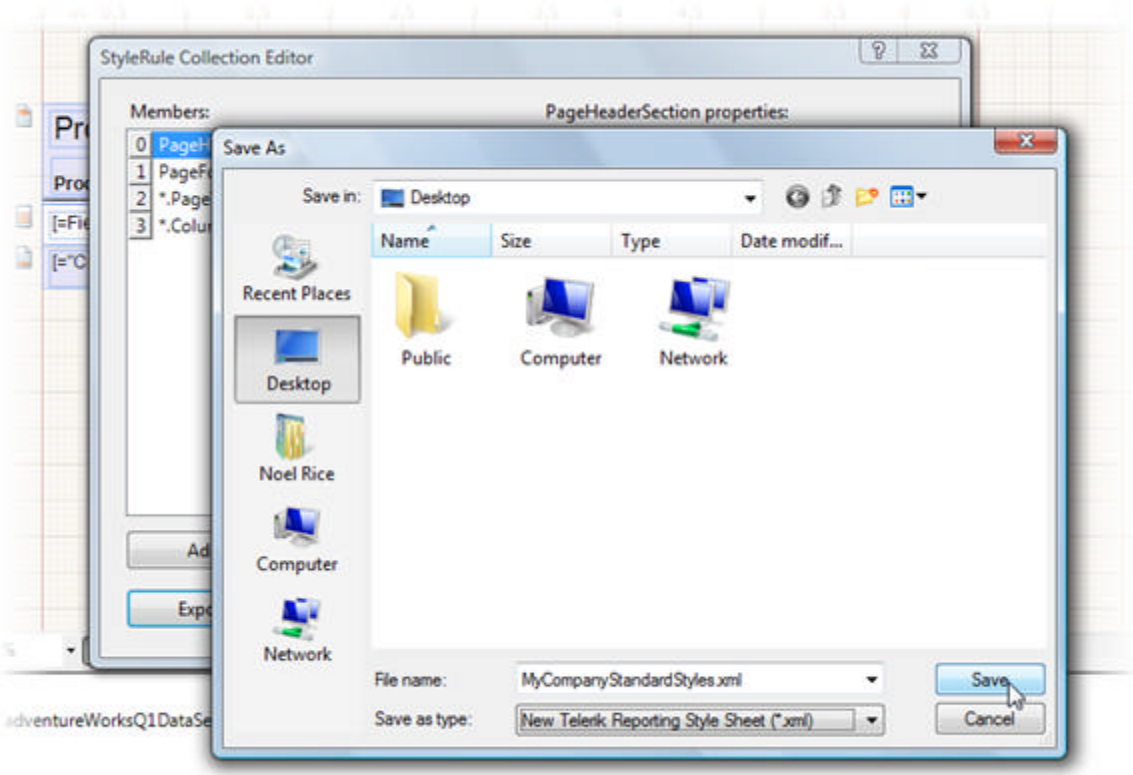
Lab: Exporting and Importing Style Rules

No one wants to repeat the same styling work on each report, even when the basic settings stay the same. You also may not want everyone putting their own "creative" stamp on every report instead of developing a company standard. And lastly, you may want someone with design experience creating the report layout, but not creating the individual reports. To satisfy all these reasons, you can export your style sheet as an XML file and import it to other reports.

Exporting

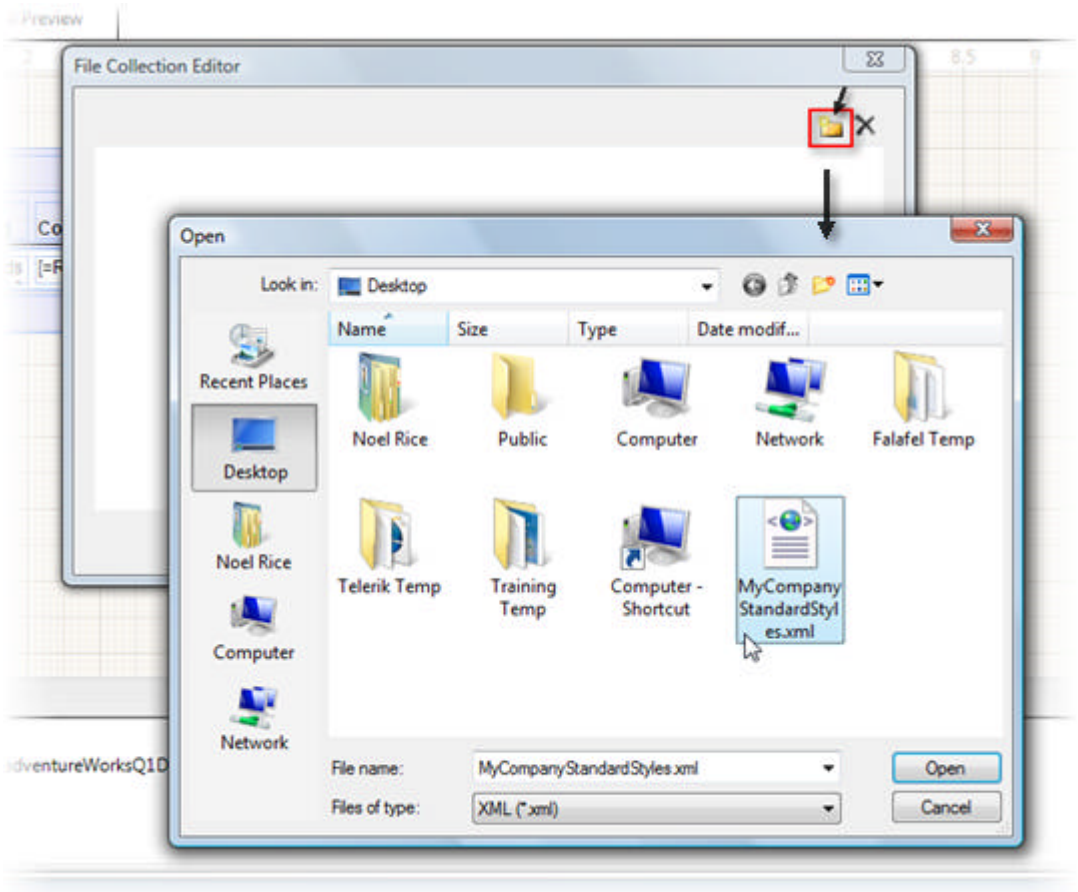
Take the previous "Adding Style Rules" example as a starting point:

1. Click the report **StyleSheet** property ellipses.
2. Click the **Export** button.
3. Provide a file name and click the **Save** button.

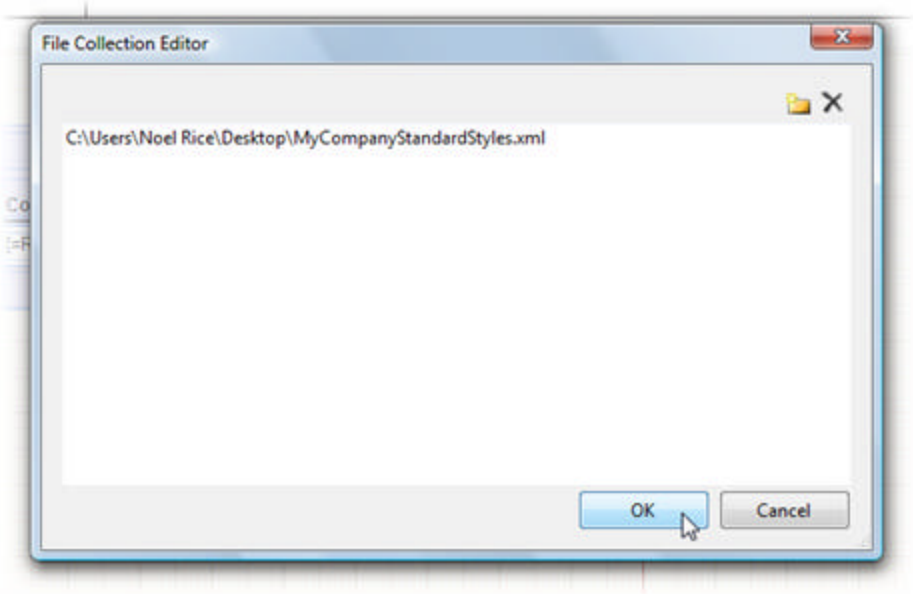


Importing

1. Click the **ExternalStyles** property ellipses. This will display a File Collection Editor dialog.
2. Click the **Add** button (found in the upper right of the dialog) to display the Open dialog. Note: you can add multiple style files to your report.
3. Select an XML file that contains previously exported styles and click the **Open** button.



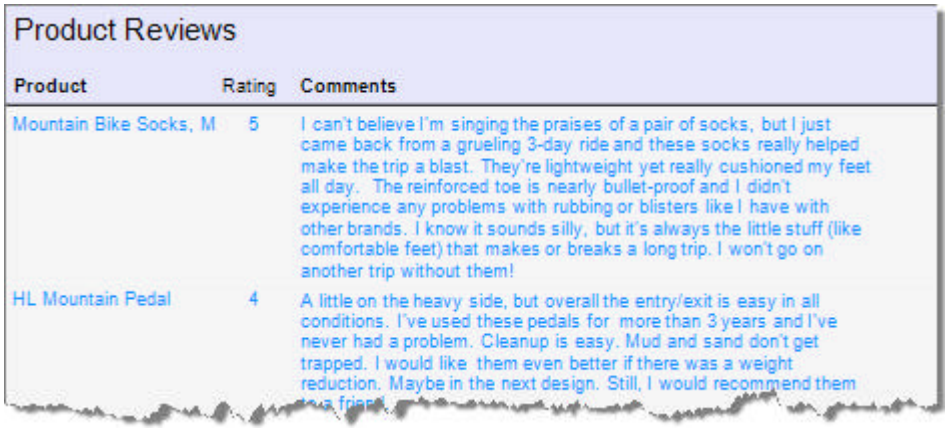
4. Click **OK** to close the File Collection Editor and see the styles applied.



Creating a StyleRule in Code

You can create the run-time counterparts to the design-time styling objects, i.e. StyleRule, selectors, styles. Add the example code below to the report constructor for the previous "Adding Style Rules" project. The code

creates a selector on the detail section of the report and applies changes to the detail section style so that the background is a light gray and the font is blue.



Product Reviews

Product	Rating	Comments
Mountain Bike Socks, M	5	I can't believe I'm singing the praises of a pair of socks, but I just came back from a grueling 3-day ride and these socks really helped make the trip a blast. They're lightweight yet really cushioned my feet all day. The reinforced toe is nearly bullet-proof and I didn't experience any problems with rubbing or blisters like I have with other brands. I know it sounds silly, but it's always the little stuff (like comfortable feet) that makes or breaks a long trip. I won't go on another trip without them!
HL Mountain Pedal	4	A little on the heavy side, but overall the entry/exit is easy in all conditions. I've used these pedals for more than 3 years and I've never had a problem. Cleanup is easy. Mud and sand don't get trapped. I would like them even better if there was a weight reduction. Maybe in the next design. Still, I would recommend them to a friend.

[C#] Adding a StyleRule

```
//Create a StyleRule
Telerik.Reporting.Drawing.StyleRule myStyleRule =
       new Telerik.Reporting.Drawing.StyleRule();
//Add a TypeSelector
myStyleRule.Selectors.Add(
   new Telerik.Reporting.Drawing.TypeSelector(
       typeof(Telerik.Reporting.DetailSection)));
//Add formatting
myStyleRule.Style.BackgroundColor = System.Drawing.Color.WhiteSmoke;
myStyleRule.Style.Color = System.Drawing.Color.DodgerBlue;
//Add rule to Style Sheet
this.StyleSheet.Add(myStyleRule);
```

[VB] Adding a StyleRule

```
'Create a StyleRule
Dim myStyleRule As New Telerik.Reporting.Drawing.StyleRule()
'Add a TypeSelector
myStyleRule.Selectors.Add(New Telerik.Reporting.Drawing.TypeSelector(GetType
(Telerik.Reporting.DetailSection)))
'Add formatting
myStyleRule.Style.BackgroundColor = System.Drawing.Color.WhiteSmoke
myStyleRule.Style.Color = System.Drawing.Color.DodgerBlue
'Add rule to Style Sheet
Me.StyleSheet.Add(myStyleRule)
```

Summary

In this section you learned how to work with styles, how to create style rules, how to export stylesheets and how to re-use stylesheets in other reports.

13 Conditional Formatting

Objectives

This section demonstrates how to perform common useful tasks such as displaying alternating row colors and highlighting report items based on data conditions using conditional formatting.

Conditional Formatting Basics

Conditional Formatting lets you react to report conditions in a visual way. You can display overdue balances in bold font, expired credit cards in red font, shade a section gray to indicate a record is past a certain date or alternate shading on every other line. Conditional Formatting lets you handle a wide variety of situations with a consistent mechanism and minimal re-learning as new situations arise. In most cases, Conditional Formatting will replace responding to events in code.

Each report item, including the report itself and each individual section, has a **ConditionalFormatting** property. The **ConditionalFormatting** property contains a collection of formatting rules that are evaluated in the order they are defined. Each rule has an associated style that lets you change the appearance of an item when the rule is satisfied. You can choose to stop evaluating rules if a particular rule condition is met.

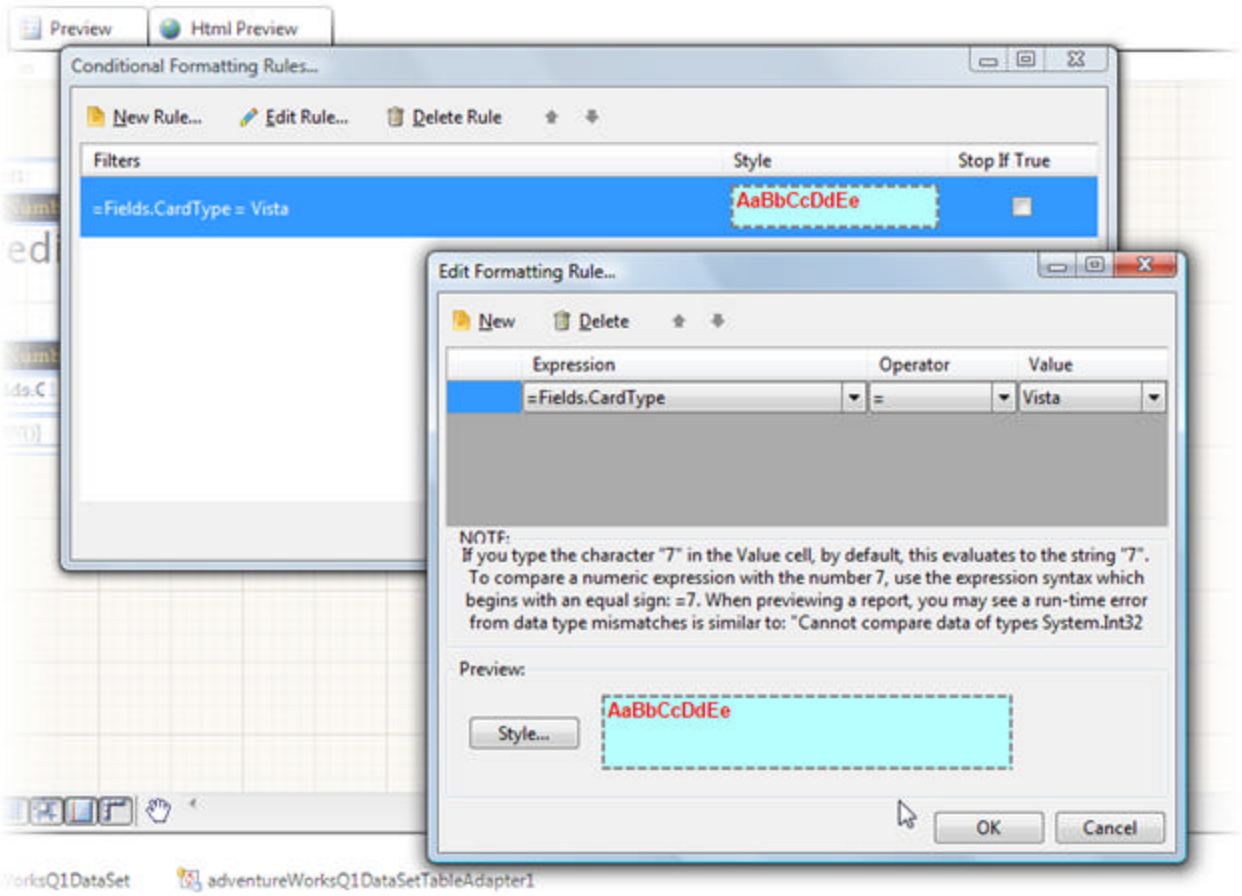
Let's take a simple example using a list of credit cards where we want to highlight a fictional credit card type called "Vista".

Credit Cards

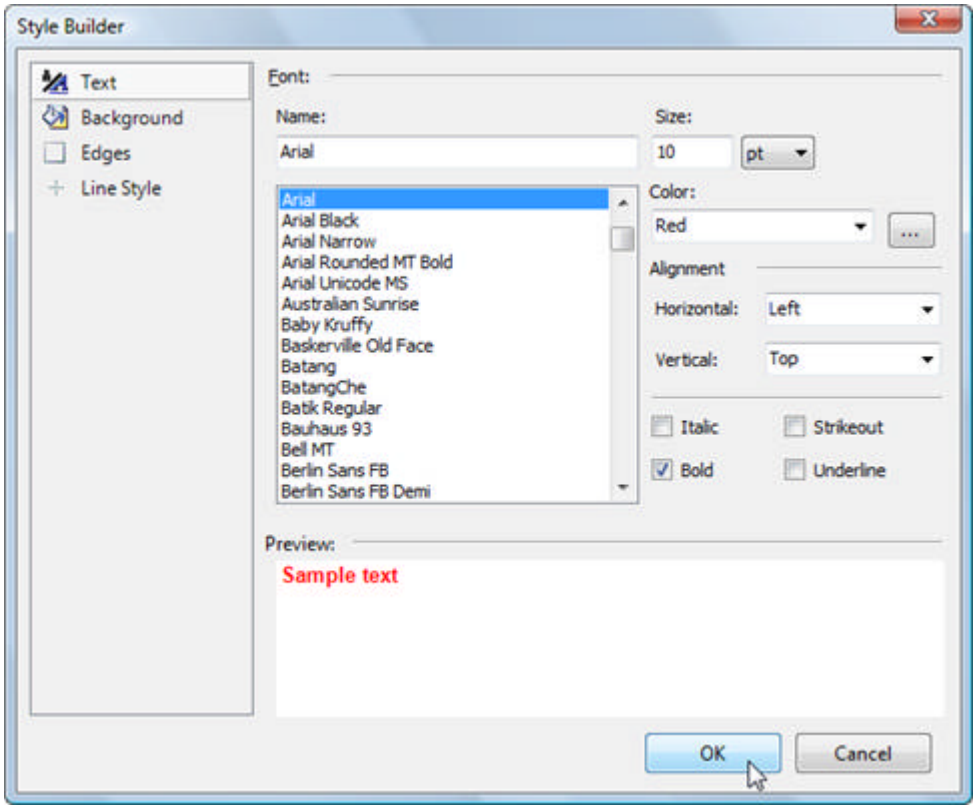
CardNumber	CardType
33332664695310	SuperiorCard
55552127249722	Distinguish
77778344838353	ColonialVoice
77774915718248	ColonialVoice
11114404600042	Vista
55557132036181	Distinguish
55553635401028	Distinguish
33336081193101	SuperiorCard
55553465625901	Distinguish
33332126386493	SuperiorCard
33335352517363	SuperiorCard
33334316194519	SuperiorCard
11119775847802	Vista
55553287727410	Distinguish
33336866065599	SuperiorCard
11111985451507	Vista

In this situation we only want to format the TextBox report item that contains the credit card type. You can select the TextBox in the designer and use the ConditionalFormatting property to invoke the **Conditional Formatting Rules...** dialog. From there you can define formatting rules that describe the condition that triggers formatting and the format itself. The example below shows formatting that is triggered when

'Fields.CardType = "Vista"'.

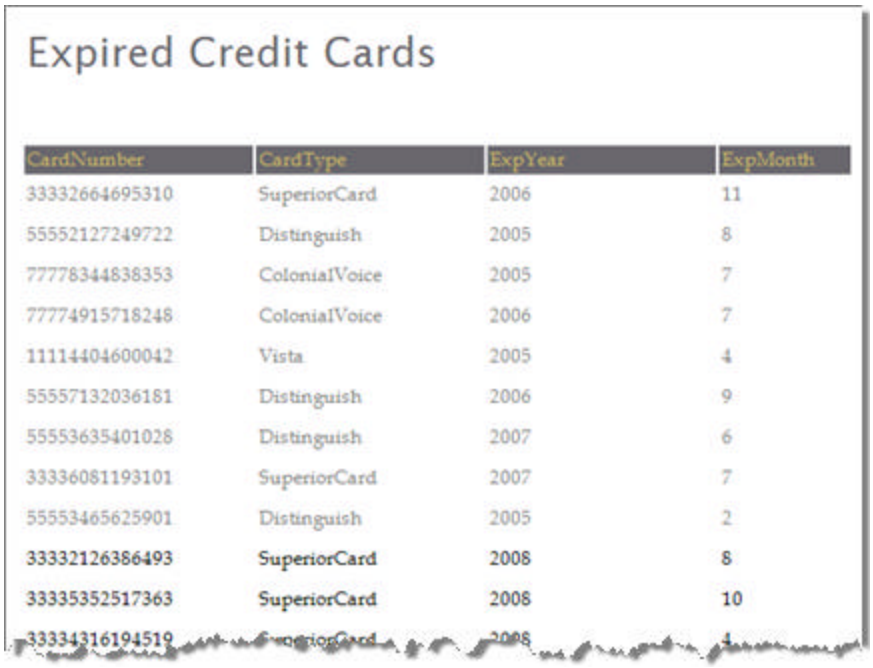


Filters are made up of one or more filter conditions. Each filter condition is made up of an **Expression**, **Operator** and **Value**. When all filter conditions are met, the rule is fired and the style is applied. To style the report section or report item, click the **Style...** button to invoke the **Style Builder** dialog. Here you can change the Text, Background, Edges and Line Style.



Formatting Rows

You can format the entire row by setting the ConditionalFormatting property of the detail section. There you can check the value of any bound data column and set the format based on the data. Using the credit card list example, we can "gray-out" rows where the credit card has expired.



Expired Credit Cards

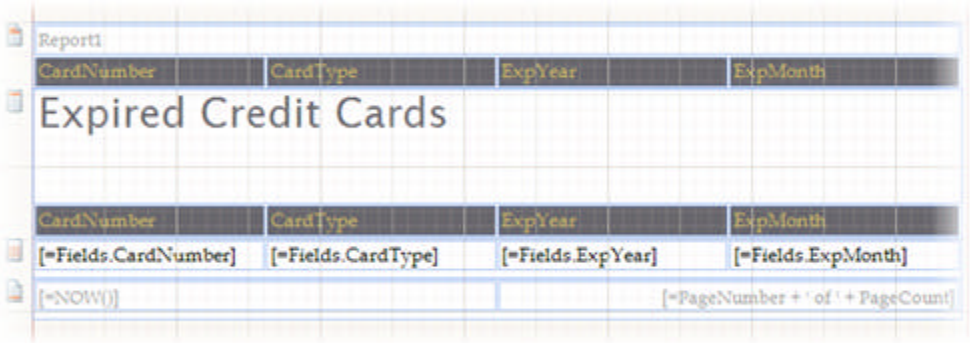
CardNumber	CardType	ExpYear	ExpMonth
33332664695310	SuperiorCard	2006	11
55552127249722	Distinguish	2005	8
77778344838353	ColonialVoice	2005	7
77774915718248	ColonialVoice	2006	7
11114404600042	Vista	2005	4
55557132036181	Distinguish	2006	9
55553635401028	Distinguish	2007	6
33336081193101	SuperiorCard	2007	7
55553465625901	Distinguish	2005	2
33332126386493	SuperiorCard	2008	8
33335352517363	SuperiorCard	2008	10
33334316194519	SuperiorCard	2008	4

1. Create a new Report class.
2. Using the Report Wizard, create a simple listing of credit cards using the SQL below against the AdventureWorks database. Select all columns from the query for display in the report.

   [SQL] List Credit Cards

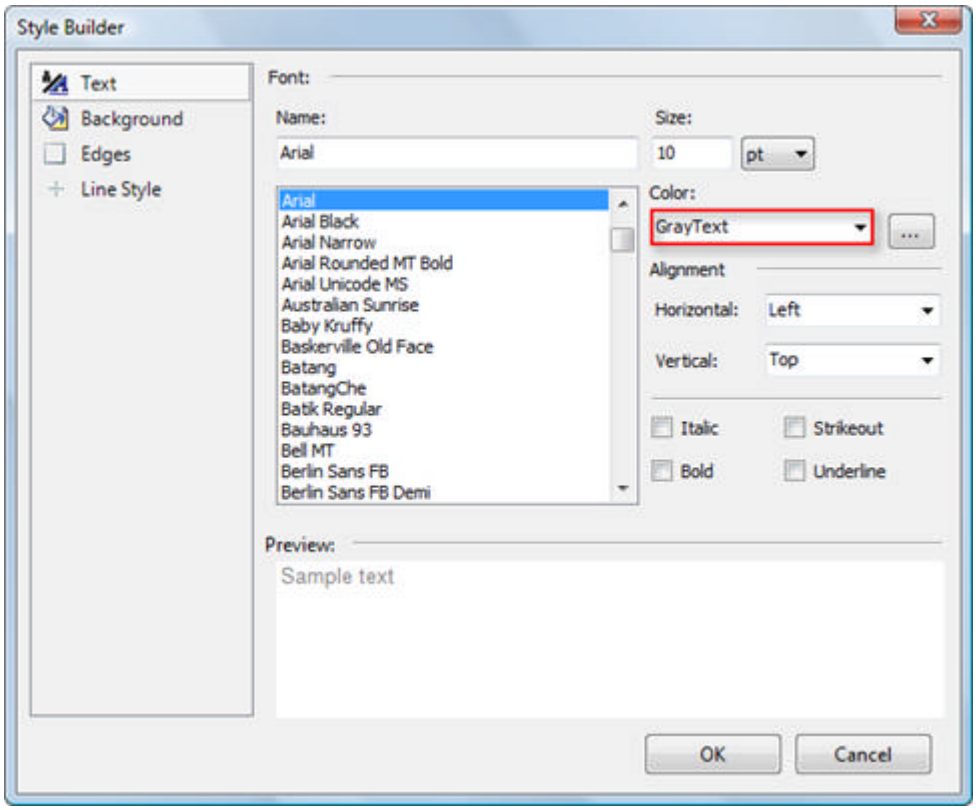
   ```
   SELECT TOP (100) CardNumber, CardType, ExpYear, ExpMonth
   FROM Sales.CreditCard
   ```

   The report will look something like the screen shot below (this example uses the "Apex" style, but you can use any of the available styles):



3. Select the detail section of the report.
4. In the Properties Window, locate the ConditionalFormatting property and click the ellipses. This step will invoke the **Conditional Formatting Rules...** dialog.
5. Click the **New Rule...** button. This step will invoke the **Edit Formatting Rule** dialog.
6. Click the **New** button to create a new condition and format.
7. In the **Expression** column enter the expression below. *The expression creates a date suitable for comparing with a system date. The date is created from the year and month found in the credit card table*:

   =CDate(CStr(Fields.ExpMonth) + "/" + "01" + "/" + Cstr(Fields.ExpYear))
8. In the **Operator** column select "<" from the drop down list.
9. In the **Value** column enter the expression "= Now()".
10. Click the **Style** button. This step will invoke the Style Builder dialog.
11. Click **Text** from the list on the left side of the Style Builder dialog. Select "GrayText" from the Color drop down list.

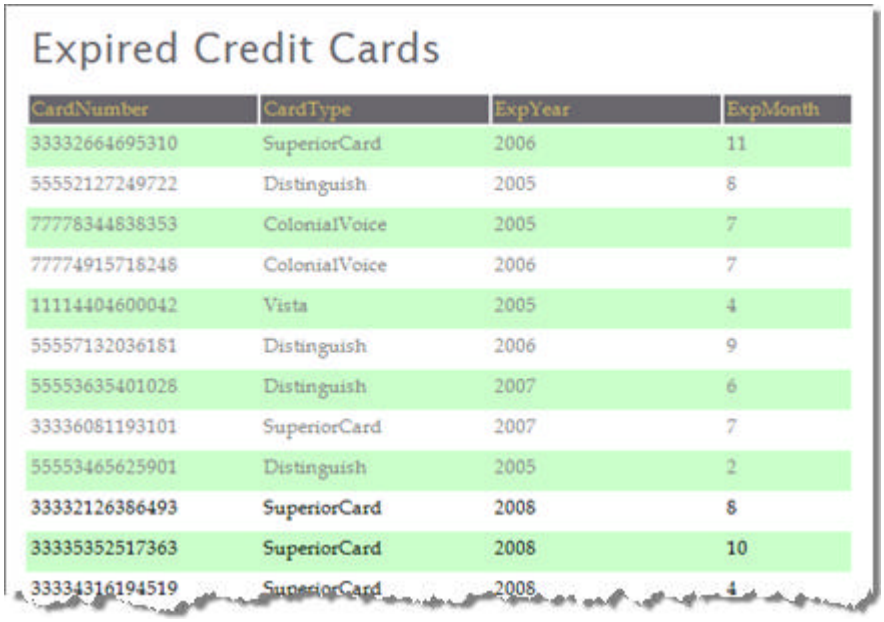


12. Click **OK** to close the Style Builder dialog.
13. Click **OK** to close the Edit Formatting Rule... dialog.
14. Click **OK** to close the Conditional Formatting Rules dialog.
15. Click the **Preview** tab to see the report.

> Depending on the current value of "Now()", you may not be seeing enough contrast in the data (in other words, all the dates are expired). Instead of "Now()" you can use another expression with a hard-coded date for this exercise, e.g. "= CDate("1/1/2008")"

Alternating Row Styles

Production-level, business reports typically use alternating row styles for better readability. You can extend the credit card report to incorporate this look:



Expired Credit Cards

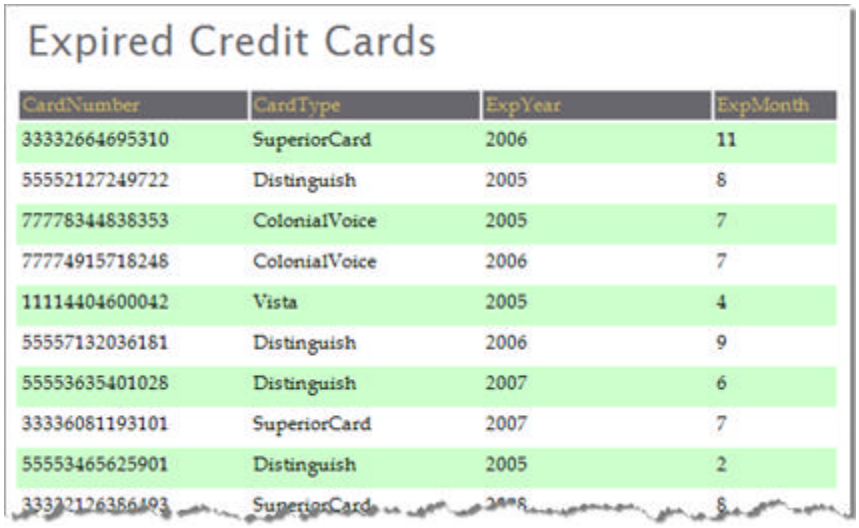
CardNumber	CardType	ExpYear	ExpMonth
33332664695310	SuperiorCard	2006	11
55552127249722	Distinguish	2005	8
77778344838353	ColonialVoice	2005	7
77774915718248	ColonialVoice	2006	7
11114404600042	Vista	2005	4
55557132036181	Distinguish	2006	9
55553635401028	Distinguish	2007	6
33336081193101	SuperiorCard	2007	7
55553465625901	Distinguish	2005	2
33332126386493	SuperiorCard	2008	8
33335352517363	SuperiorCard	2008	10
33334316194519	SuperiorCard	2008	4

1. Select the detail section of your report.
2. Select the **ConditionalFormatting** property of the detail section.
3. Click the **New Rule** button.
4. Enter the expression "= RowNumber()%2"
5. Select the equal "=" operator from the drop down list.
6. Enter the value as "=1".
7. Click the **Style** button.
8. Click the **Background** style item.
9. Enter the color as "#CAFFCA".
10. Click **OK** to close each of the open dialogs.
11. Click the **Preview** tab to view the report.

Adding Conditional Formatting Programmatically

We can match the effect of any conditional formatting at design time in code by adding **FormattingRule** objects to the ConditionalFormatting collection for any report item. Like it's design-time counterpart, FormattingRule has a **Filters** collection and a **Style** property.

Let's take the credit card example and set the alternate row formatting in code. First, take your previous credit card report example and delete the current conditional formatting rules. Then place the code below in the constructor. The report should look something like the screenshot:



Expired Credit Cards

CardNumber	CardType	ExpYear	ExpMonth
33332664695310	SuperiorCard	2006	11
55552127249722	Distinguish	2005	8
77778344838353	ColonialVoice	2005	7
77774915718248	ColonialVoice	2006	7
11114404600042	Vista	2005	4
55557132036181	Distinguish	2006	9
55553635401028	Distinguish	2007	6
33336081193101	SuperiorCard	2007	7
55553465625901	Distinguish	2005	2
33332126386493	SuperiorCard	2006	8

[C#] Define and Apply Conditional Formatting Rules

```
// Note: this example requires the Telerik.Reporting.Data namespace reference
// Define an alternating row style
FormattingRule alternateRowsRule = new FormattingRule();
// Define the rule to trigger the format
alternateRowsRule.Filters.Add(
   new Filter("= RowNumber()%2", FilterOperator.Equal, "=1"));
// Define the style to be applied when rule is true
alternateRowsRule.Style.BackgroundColor = Color.FromArgb(204, 255, 204);
// Add the rule to the ConditionalFormatting collection
this.detail.ConditionalFormatting.Add(alternateRowsRule);
```

[VB] Define and Apply Conditional Formatting Rules

```
' Note: this example requires the Telerik.Reporting.Data namespace reference
' Define an alternating row style
Dim alternateRowsRule As New FormattingRule()
' Define the rule to trigger the format
alternateRowsRule.Filters.Add(New Filter("= RowNumber()%2", FilterOperator.Equal, "=1"))
' Define the style to be applied when rule is true
alternateRowsRule.Style.BackgroundColor = Color.FromArgb(204, 255, 204)
' Add the rule to the ConditionalFormatting collection
Me.detail.ConditionalFormatting.Add(alternateRowsRule)
```

Summary

In this section you learned how to apply conditional formatting to react to different kinds of data and to perform common tasks like alternating row colors.

14 Reports at Runtime

Objectives

- Learn to create a report section using code only.
- Learn to create a report item using code only.
- Create TextBox, Shape and PictureBox report items at run-time.

Lab: Create Report Sections and Items

A Telerik report is a .NET class the same as any other, so you can create an instance of that class and set its properties. Actually, the report designer does nothing more than create items and set properties for you. So, the best way to learn how to create a report programmatically is to use the designer to create a report and then look at the code generated by the designer.

Creating Sections

To get started, create a new report class and cancel the Report Wizard. You should have three empty report sections, i.e. page header, detail and page footer. If you look at the generated code found in "Report1.Designer.cs" in the region titled "Component Designer Generated Code" for the detail section you see:

[C#] Generated Detail Section Code

```
this.detail = new Telerik.Reporting.DetailSection();
//. . .
this.detail.Height = new Telerik.Reporting.Drawing.Unit(0.21045596897602081,
((Telerik.Reporting.Drawing.UnitType)(Telerik.Reporting.Drawing.UnitType.Inch)));
this.detail.Name = "detail";
```

[VB] Generated Detail Section Code

```
Me.detail = New Telerik.Reporting.DetailSection()
'. . .
Me.detail.Height = New Telerik.Reporting.Drawing.Unit(0.210455968976021, (DirectCast
((Telerik.Reporting.Drawing.UnitType.Inch), Telerik.Reporting.Drawing.UnitType)))
Me.detail.Name = "detail"
```

In this same way you can create new sections for all portions of the report. Here are the objects that encapsulate the functionality of each type of section: **DetailSection**, **GroupHeaderSection**, **GroupFooterSection**, **PageHeaderSection**, **PageFooterSection**, **ReportHeader** and **ReportFooterSection**. The general pattern is that the section is created, assigned to the appropriate section property of the report and the section properties are defined.

Creating Report Items

Likewise, report items are created in much the same way. Drop a TextBox control in the detail section of the report and again check the generated code:

[C#] Generated Code Defining a TextBox

```
this.textBox1 = new Telerik.Reporting.TextBox();
// ...
this.detail.Items.AddRange(new Telerik.Reporting.ReportItemBase[] {
this.textBox1});
//...
this.textBox1.Location = new Telerik.Reporting.Drawing.PointU(new
Telerik.Reporting.Drawing.Unit(0.19999997317790985, ((Telerik.Reporting.Drawing.UnitType)
(Telerik.Reporting.Drawing.UnitType.Inch))), new Telerik.Reporting.Drawing.Unit
```

```
(0.010456006042659283, ((Telerik.Reporting.Drawing.UnitType)
(Telerik.Reporting.Drawing.UnitType.Inch))));
this.textBox1.Name = "textBox1";
this.textBox1.Size = new Telerik.Reporting.Drawing.SizeU(new Telerik.Reporting.Drawing.Unit
(0.7999998927116394, ((Telerik.Reporting.Drawing.UnitType)
(Telerik.Reporting.Drawing.UnitType.Inch))), new Telerik.Reporting.Drawing.Unit
(0.19999997317790985, ((Telerik.Reporting.Drawing.UnitType)
(Telerik.Reporting.Drawing.UnitType.Inch))));
this.textBox1.Value = "textBox1";
```

[VB] Generated Code Defining a TextBox

```
Me.textBox1 = New Telerik.Reporting.TextBox()
' ...
Me.detail.Items.AddRange(New Telerik.Reporting.ReportItemBase() {Me.textBox1})
'...
Me.textBox1.Location = New Telerik.Reporting.Drawing.PointU(New
Telerik.Reporting.Drawing.Unit(0.19999997317791, (DirectCast
((Telerik.Reporting.Drawing.UnitType.Inch), Telerik.Reporting.Drawing.UnitType))), New
Telerik.Reporting.Drawing.Unit(0.0104560060426593, (DirectCast
((Telerik.Reporting.Drawing.UnitType.Inch), Telerik.Reporting.Drawing.UnitType))))
Me.textBox1.Name = "textBox1"
Me.textBox1.Size = New Telerik.Reporting.Drawing.SizeU(New Telerik.Reporting.Drawing.Unit
(0.799999892711639, (DirectCast((Telerik.Reporting.Drawing.UnitType.Inch),
Telerik.Reporting.Drawing.UnitType))), New Telerik.Reporting.Drawing.Unit(0.19999997317791,
(DirectCast((Telerik.Reporting.Drawing.UnitType.Inch),
Telerik.Reporting.Drawing.UnitType))))
Me.textBox1.Value = "textBox1"
```

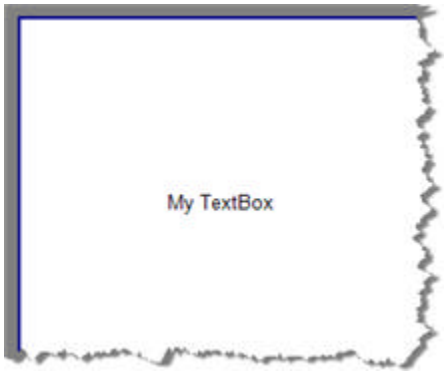
Report Item Examples

TextBox

The minimum that must be done to make a TextBox report item visible in a section is:

- Create the TextBox Instance
- Assign the Value property
- Place the TextBox somewhere on the report
- Add the TextBox to a report section

For example:



[C#] Minimal Code for Creating a TextBox

```
// Create the report item instance
Telerik.Reporting.TextBox textBox = new Telerik.Reporting.TextBox();
// Assign some content that will be visible
textBox.Value = "My TextBox";
// Set the location and dimensions of the report item
textBox.Dock = DockStyle.Fill;
// Add the report item to the Items collection of a section
this.detail.Items.Add(textBox);
```

[VB] Minimal Code for Creating a TextBox

```
' Create the report item instance
Dim textBox As New Telerik.Reporting.TextBox()
' Assign some content that will be visible
textBox.Value = "My TextBox"
' Set the location and dimensions of the report item
textBox.Dock = DockStyle.Fill
' Add the report item to the Items collection of a section
Me.detail.Items.Add(textBox)
```

A common problem when dynamically creating report items is that the item is not visible, even though the item has been instantiated and added to a section of the report. ***You must set the item dimensions***. Either set the **Width** and **Height** properties directly or use the **Dock** property so that Telerik Reporting handles those details for you. If you leave out the **Dock** property assignment in the example above, the TextBox will not appear. Here's alternate code that fixes the problem by explicitly setting dimensions:

[C#] Alternate Code That Explicitly Sets Dimensions

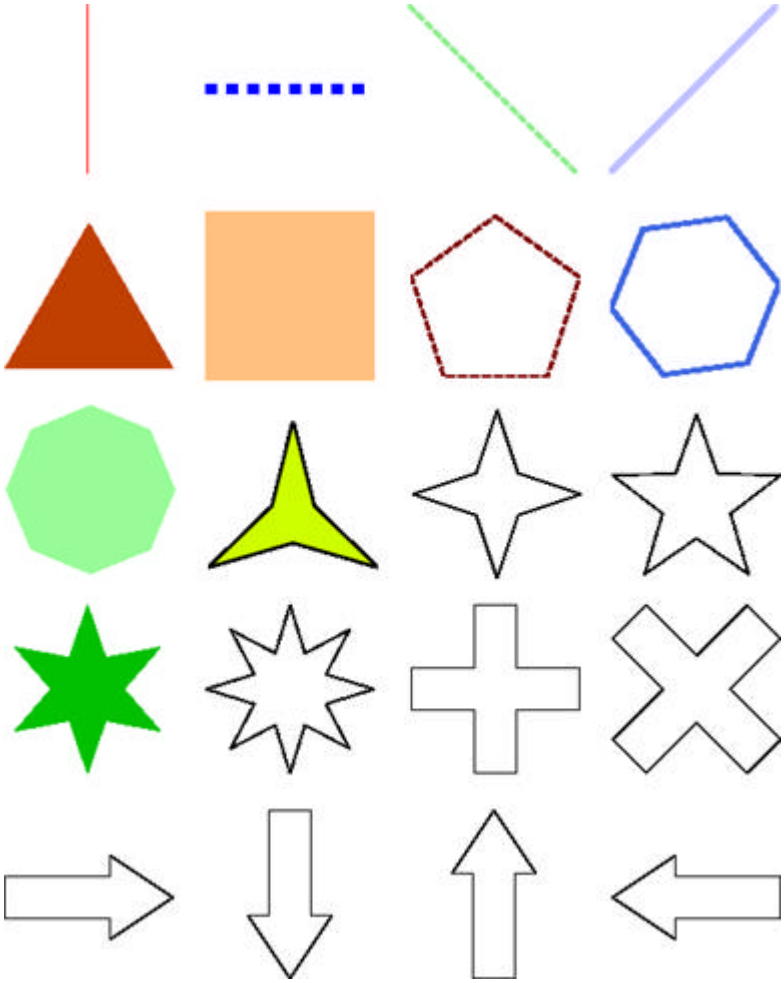
```
// Create the report item instance
Telerik.Reporting.TextBox textBox = new Telerik.Reporting.TextBox();
// Assign some content that will be visible
textBox.Value = "My TextBox";
// Set the location and dimensions of the report item
//textBox.Dock = DockStyle.Fill;
textBox.Width = new Unit(1, UnitType.Inch);
textBox.Height = new Unit(1, UnitType.Inch);
// Add the report item to the Items collection of a section
this.detail.Items.Add(textBox);
```

[VB] Alternate Code That Explicitly Sets Dimensions

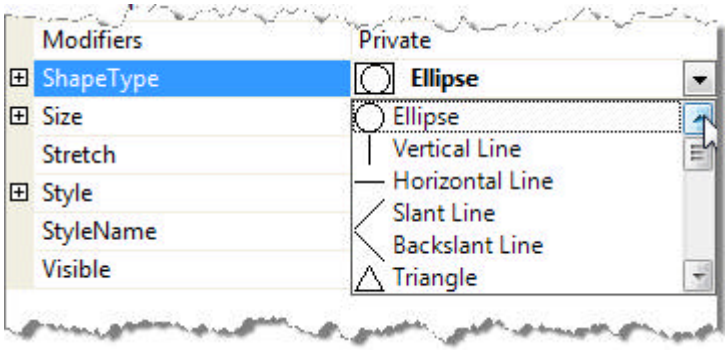
```
' Create the report item instance
Dim textBox As New Telerik.Reporting.TextBox()
' Assign some content that will be visible
textBox.Value = "My TextBox"
' Set the location and dimensions of the report item
'textBox.Dock = DockStyle.Fill;
textBox.Width = New Unit(1, UnitType.Inch)
textBox.Height = New Unit(1, UnitType.Inch)
' Add the report item to the Items collection of a section
Me.detail.Items.Add(textBox)
```

Shape

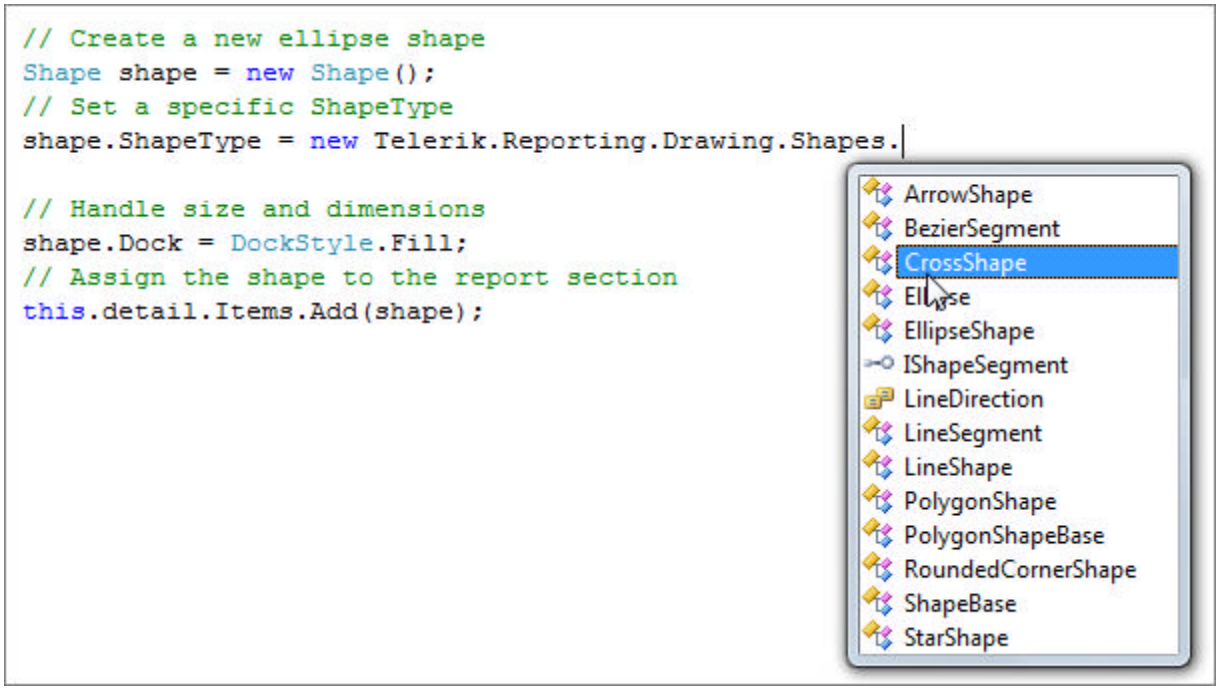
Shape report items come with a number of predefined primitive elements that can help you build visual design features of your report. Here is a sample of some of the basic shapes and how they can be styled:



Creating a Shape report item is similar to the TextBox except that you must assign a **ShapeType** instead of a Value. ShapeType takes an instance of one of the many ShapeBase descendents. In the designer you get a visual clue from the ShapeType property drop down list.



When coding you can get a list of the available ShapeTypes using Intellisense against the **Telerik.Reporting.Drawing.Shapes** namespace, with the naming convention "<a shape name>Shape".



Here is the minimal code that creates an elliptical shape in the detail section of the report.

[C#] Minimal Code for Creating a Shape

```
// Create a new ellipse shape
Shape shape = new Shape();
// Set a specific ShapeType
shape.ShapeType = new EllipseShape();
// Handle size and dimensions
shape.Dock = DockStyle.Fill;
// Assign the shape to the report section
this.detail.Items.Add(shape);
```

[VB] Minimal Code for Creating a Shape

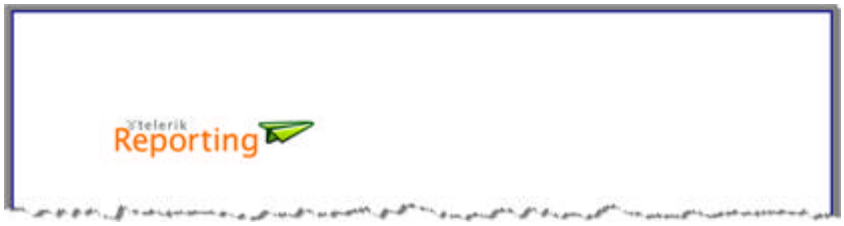
```
' Create a new ellipse shape
Dim shape As New Shape()
' Set a specific ShapeType
shape.ShapeType = New EllipseShape()
' Handle size and dimensions
shape.Dock = DockStyle.Fill
' Assign the shape to the report section
Me.detail.Items.Add(shape)
```

PictureBox

You can include images to your reports using the PictureBox report item. The critical properties here are:

- **Value:** Create an Image object and assign to this property
- **Sizing**: This adjusts how the image fits to the PictureBox, if the PictureBox fits to the image, and if any clipping should occur

The example below assigns a standard .NET framework Bitmap object to the PictureBox Value property. The image itself is the Telerik Reporting logo, taken from a Telerik web site page and added to the project.



[C#] Minimal Code for Creating a PictureBox

```
// provide enough room in the header to contain the full image
this.pageHeader.Height = new Unit(1, UnitType.Inch);
// create an instance of the PictureBox
Telerik.Reporting.PictureBox pictureBox = new Telerik.Reporting.PictureBox();
// assign a Bitmap object to the Value property
pictureBox.Value = new Bitmap("productLogo.gif");
// use AutoSize so that report item fits the image
pictureBox.Sizing = ImageSizeMode.AutoSize;
// add the report item to the page header section
this.pageHeader.Items.Add(pictureBox);
```

[VB] Minimal Code for Creating a PictureBox

```
' provide enough room in the header to contain the full image
Me.pageHeader.Height = New Unit(1, UnitType.Inch)
' create an instance of the PictureBox
Dim pictureBox As New Telerik.Reporting.PictureBox()
' assign a Bitmap object to the Value property
pictureBox.Value = New Bitmap("productLogo.gif")
' use AutoSize so that report item fits the image
pictureBox.Sizing = ImageSizeMode.AutoSize
' add the report item to the page header section
Me.pageHeader.Items.Add(pictureBox)
```

At design time you use the **Value** property to specify the PictureBox image data. The **Value** property is a **System.Object** and accepts only 2 specific types:

- **System.String** - interpreted as an item binding expression that should evaluate to **System.Drawing.Image** or **byte[].** The Image or byte array is used internally as Image data.
- **System.Drawing.Image** - the Image to be rendered.

To use static images (not from the data source) you can apply any standard NET Framework approach to load them:

- System.Drawing.Image.FromFile() loads an image from a file
- System.Drawing.Image.FromStream() loads an image from a data stream
- System.Resources.ResourceManager lets you work with embedded resources
- System.Drawing.Bitmap(Type, string) constructor loads an image from a resource

Lab: Create a Section

The example below creates a new **ReportHeaderSection** in code, but the pattern is the same for all section types. You can set the **Height** property of the section, but not the width (width is inferred from the width of

the report). Change the appearance of the section using the **Style** property. Finally, you must add the section to the report **Items** collection.

1. Using the empty report created in the previous lesson, remove any report items from the report sections.
2. Add the code below to your report class constructor:

   [C#] Creating a Report Header Section

   ```
   ReportHeaderSection headerSection = new ReportHeaderSection();
   headerSection.Height =
      new Telerik.Reporting.Drawing.Unit(.6, Telerik.Reporting.Drawing.UnitType.Inch);
   headerSection.Style.BackgroundColor = Color.LightSteelBlue;
   headerSection.Style.BorderStyle.Default = BorderType.Ridge;
   this.Items.Add(headerSection);
   ```

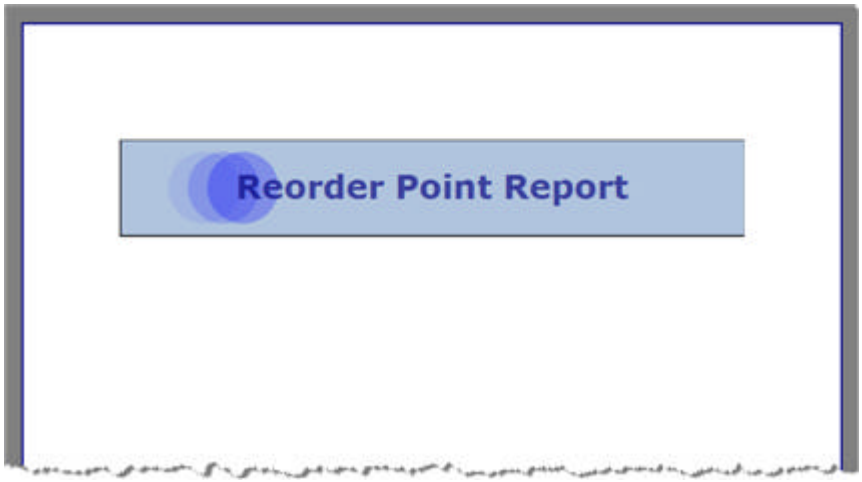
   [VB] Creating a Report Header Section

   ```
   Dim headerSection As New ReportHeaderSection()
   headerSection.Height = New Telerik.Reporting.Drawing.Unit(0.6,
   Telerik.Reporting.Drawing.UnitType.Inch)
   headerSection.Style.BackgroundColor = Color.LightSteelBlue
   headerSection.Style.BorderStyle.[Default] = BorderType.Ridge
   Me.Items.Add(headerSection)
   ```

3. Click the report designer **Preview** tab to see the result. The code example will produce a report header section that looks like this screen shot:

Lab: Create Report Items

To extend the previous report header section example we will create report items and add them to the page header section. This example adds a series of elliptical **Shape** report items and a **TextBox** to the report header. The shapes are placed from left to right, with varying transparency. The **TextBox** report item is placed over the circles to function as the title.



1. To get this effect, start with the previous "Create a Section" project and make these changes:
2. Add **Telerik.Reporting.Drawing.Shapes** to your "using" (C#) or "Imports" (VB) portion of the code.
3. Add code below to create the new report items to your report class constructor, just below the section header creation code. The code example will show the entire report class constructor. Look for the comments that indicate the start and end of the new code to be added:

[C#] Create Shapes and TextBoxes

```
public Report1()
{
   InitializeComponent();
   // Define header section, set dimensions/appearance, and add to report
   ReportHeaderSection headerSection = new ReportHeaderSection();
   headerSection.Height =
       new Telerik.Reporting.Drawing.Unit(1, UnitType.Inch);
   headerSection.Style.BackgroundColor = Color.LightSteelBlue;
   headerSection.Style.BorderStyle.Default = BorderType.Ridge;
   this.Items.Add(headerSection);
   // --- Start new code ---
   // Unit size used for both height and width
   Unit shapeSize = new Unit(.75, UnitType.Inch);
   // variable to track left-to-right placement of circle shapes
   double currentX = .5;
   // variable to track transparency of circle shapes
   int currentAlpha = 20;
   // controls the number of iterations when creating circle shapes
   const int numberOfCircles = 3;
   // amount to increment horizontal placement of circle shapes
   const double addToX = .2;
   // amount to increment alpha
   const int addToAlpha = 25;

   // create a series of circles behind the text
   for (int i = 0; i < numberOfCircles; i++)
   {
       // create a new ellipse shape and set the dimensions
       Shape shape = new Shape();
       shape.ShapeType = new EllipseShape();
       shape.Height = shapeSize;
```

```
        shape.Width = shapeSize;
        // set the shape location and color. The shape should be centered vertically and
placed on the
        // "currentX" horizontally. Color is assigned from the shapeColors array defined
above.
        double shapeOffset = (headerSection.Height.Value - shape.Height.Value) / 2;
        shape.Location = new PointU(new Unit(currentX, UnitType.Inch), new Unit(shapeOffse
UnitType.Inch));
        // set the shape background and border color - each iteration the color becomes
darker, more opaque
        shape.Style.BackgroundColor = Color.FromArgb(currentAlpha, Color.Blue);
        shape.Style.Color = Color.FromArgb(currentAlpha, Color.LightBlue);
        // add the circle to the header Items collection
        headerSection.Items.Add(shape);
        // move the X coordinate to the right
        currentX += addToX;
        // make alpha more opaque
        currentAlpha += addToAlpha;
    }

    // Create title text box and add to the header section
    // Note the use of Dock, TextAlign and VerticalAlign to place
    // the title in the center of the header.
    Telerik.Reporting.TextBox textBox = new Telerik.Reporting.TextBox();
    textBox.Value = "Reorder Point Report";
    textBox.Style.Font.Size = new Unit(25, UnitType.Point);
    textBox.Style.Font.Bold = true;
    textBox.Style.Font.Name = "Verdana";
    textBox.Style.Color = Color.DarkBlue;
    textBox.Dock = DockStyle.Fill;
    textBox.Style.TextAlign = HorizontalAlign.Center;
    textBox.Style.VerticalAlign = VerticalAlign.Middle;
    headerSection.Items.Add(textBox);
    // --- End new code ---
}
```

[VB] Create Shapes and TextBoxes

```
Public Sub New()
 InitializeComponent()
 ' Define header section, set dimensions/appearance, and add to report
 Dim headerSection As New ReportHeaderSection()
 headerSection.Height = New Telerik.Reporting.Drawing.Unit(1, UnitType.Inch)
 headerSection.Style.BackgroundColor = Color.LightSteelBlue
 headerSection.Style.BorderStyle.[Default] = BorderType.Ridge
 Me.Items.Add(headerSection)
 ' --- Start new code ---
 ' Unit size used for both height and width
 Dim shapeSize As New Unit(0.75, UnitType.Inch)
 ' variable to track left-to-right placement of circle shapes
 Dim currentX As Double = 0.5
 ' variable to track transparency of circle shapes
 Dim currentAlpha As Integer = 20
 ' controls the number of iterations when creating circle shapes
 Const numberOfCircles As Integer = 3
```

```
 ' amount to increment horizontal placement of circle shapes
 Const addToX As Double = 0.2
 ' amount to increment alpha
 Const addToAlpha As Integer = 25
 ' create a series of circles behind the text
 Dim i As Integer = 0
 While i < numberOfCircles
  ' create a new ellipse shape and set the dimensions
  Dim shape As New Shape()
  shape.ShapeType = New EllipseShape()
  shape.Height = shapeSize
  shape.Width = shapeSize
  ' set the shape location and color. The shape should be centered vertically and placed
the
  ' "currentX" horizontally. Color is assigned from the shapeColors array defined above.
  Dim shapeOffset As Double = (headerSection.Height.Value - shape.Height.Value) / 2
  shape.Location = New PointU(New Unit(currentX, UnitType.Inch), New Unit(shapeOffset,
UnitType.Inch))
  ' set the shape background and border color - each iteration the color becomes darker,
more opaque
  shape.Style.BackgroundColor = Color.FromArgb(currentAlpha, Color.Blue)
  shape.Style.Color = Color.FromArgb(currentAlpha, Color.LightBlue)
  ' add the circle to the header Items collection
  headerSection.Items.Add(shape)
  ' move the X coordinate to the right
  currentX += addToX
  ' make alpha more opaque
  currentAlpha += addToAlpha
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
 ' Create title text box and add to the header section
 ' Note the use of Dock, TextAlign and VerticalAlign to place
 ' the title in the center of the header.
 Dim textBox As New Telerik.Reporting.TextBox()
 textBox.Value = "Reorder Point Report"
 textBox.Style.Font.Size = New Unit(25, UnitType.Point)
 textBox.Style.Font.Bold = True
 textBox.Style.Font.Name = "Verdana"
 textBox.Style.Color = Color.DarkBlue
 textBox.Dock = DockStyle.Fill
 textBox.Style.TextAlign = HorizontalAlign.Center
 textBox.Style.VerticalAlign = VerticalAlign.Middle
  ' --- End new code ---
 headerSection.Items.Add(textBox)
End Sub
```

Summary

In this lesson you learned how to create report sections and items at run-time. You also became familiar with the specifics of the TextBox, Shape and PictureBox report items.

15 Events

Objectives

In this section you will also be introduced to the report life cycle - a critical part of understanding how data is made available, at what times data is available and in what sections of the report. The report, section and item events will be shown and then put in the context of the report life cycle. Finally, you will create an event handler for the report detail section that uses data from multiple columns to format a single column.

The Report Life Cycle

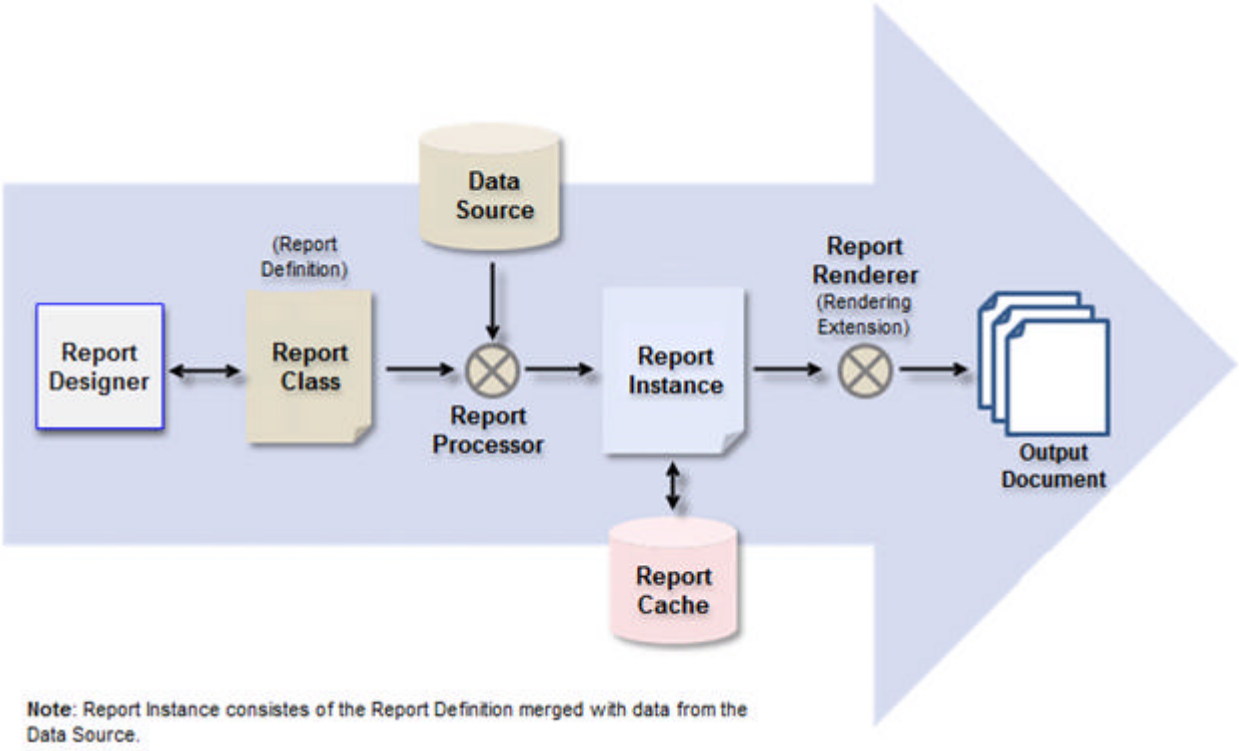
Understanding the report life cycle, what happens and when it happens is crucial to effectively using Telerik Reporting. It's particularly important to know what report sections may contain data bound fields, aggregates and global objects.

> The key point to take-away here is that paging happens late in the life cycle. This affects what you can have in the page header and footer sections: **PageNumber** and **PageCount** global objects are *only* available to the page header and footer. Likewise, data binding occurs earlier in the life cycle, so data binding and aggregate functions are *not* available to the page header and footer sections.

Why Report Lifecycle is Important

There are many ramifications of the report lifecycle including when you can use bound data fields, aggregates and page information functions.

- **Page Headers and Footers**: Item binding expressions ***cannot*** use data fields or aggregate functions. Paging is specific to rendering format and must occur during rendering, well after early report processing where data binding occurs. ***PageNumber*** and ***PageCount*** global objects are available only in this section.
- **Report Headers, Footers and Groups**: For these sections, only aggregate functions should be used in expressions. If an expression specifies a data field without an aggregate, the **First()** aggregate function is used by default. The **First()** function returns the first record of the data source for the specified field.
- **Detail Section**: The detail section can contain data bound fields and any function available in the Edit Expression dialog except PageCount and PageNumber.



Report Lifecycle Overview

1. **The report is designed** in the Visual Studio Report Designer. The act of designing the report produces a report class that embodies the Report Definition.
2. **The report is processed**:
   - Binds the Report Definition with data from the report data source.
   - Performs all grouping, sorting and filtering calculations.
   - Evaluates all Expressions except page header and footer section items.
   - Fires item Binding and Bound events.
   - Produces a Report Instance. Report instances may be stored and rendered at a later time.
3. **The report is rendered**:
   - The Report Instance is passed to a specific rendering extension (e.g. MHTML or PDF output formats).
   - The report instance is paged if the output format supports paging.
   - Item expressions are evaluated in the page header and footer sections for every page.
   - The report is rendered to the Output Document.

Rendering

Telerik reports can be rendered to a number of output formats. Each format has characteristics that determine usage and may impose certain limitations. For example, a report rendered as HTML you can expect to have a dynamic, HTML table-based arrangement while a PDF report will have an exact set of dimensions. Each output format is produced by a *rendering extension*. You can break these rendering extensions out into a few general categories:

- **Data**: CSV

- **Logical page layout**: HTML, Excel
- **Physical page layout**: Image, PDF

See the online help under "Design Considerations for..." for more detail involving extension specific rendering, pagination and potential limitations.

Also be aware that page setup options can also alter report appearance at design-time, run-time or when the user changes the options prior to printing. Page setup options are controlled by the PageSettings property and apply to all report output types.

- Landscape or Portrait paper orientation.
- Margin sizes.
- Paper Kind: This option includes standard paper size configurations e.g. legal, envelopes, labels.
- Paper Size: If the PaperKind property is "Custom", then the PaperSize Width and Height properties can be set to whatever dimensions suit your purpose.

PageSettings	Telerik.Reporting.Drawing.PageSetting
Landscape	False
⊟ Margins	Telerik.Reporting.Drawing.MarginsU
Bottom	1.00in
Left	1.00in
Right	1.00in
Top	1.00in
PaperKind	Letter
PaperSize	8.50in, 11.00in

Handling Report, Section and Item Events

Reports, sections and report items all have events for data binding and for sensing when items have been added or removed. These events provide the ultimate flexibility if you need very detailed control over some piece of the report. ***Before you use an event***, be sure that conditional formatting, expressions or user defined functions won't do the job. These tools are the most productive way to build reports and should be favored over using events.

The Report object events are:

Property	*Description*
ItemAdded	Fires when each item is added to the report.
ItemDataBinding	Fires just before each report item is bound to data. Can be canceled.
ItemDataBound	Fires just after each report item is bound to data.
ItemRemoved	Fires when an item is removed from the report.
NeedDataSource	Fires when the report does not have a valid data source bound to it.

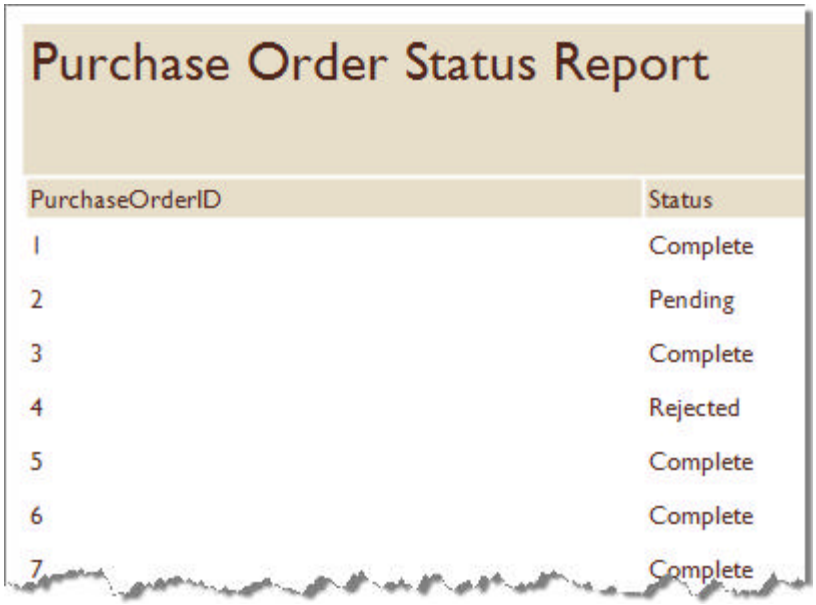
Section and report item events are:

Property	*Description*
ItemAdded	Fires when each item is added to the section.
ItemDataBinding	Fires just before the section is bound to data. Can be canceled.

ItemDataBound	Fires just after the section is bound to data.
ItemRemoved	Fires when an item is removed from the section.

> The **SubReport** item, like the Report, also has a **NeedDataSource** that fires when the subreport is not data bound.

Here is a simple example that shows a situation where you might choose an **ItemDataBound** event over an expression. Even in this situation you could use expressions to get the same effect. Depending on your infrastructure and in-house standards, you may prefer to perform some tasks using code. In this example we handle the ItemdataBound event for a TextBox that contains a status number. We have an enumeration defined that maps the numbers to meaningful status descriptions.



In the code we use the "sender" as the reference to the TextBox, get the status number and convert it to the enumeration's string representation.

[C#] Handling the ItemDataBound Event

```
enum Status { Pending = 1, Approved = 2, Rejected = 3, Complete = 4 } ;
private void statusDataTextBox_ItemDataBound(object sender, System.EventArgs e)
{
   Telerik.Reporting.Processing.TextBox txtStatus = (Telerik.Reporting.Processing.TextBox)
sender;
   txtStatus.Text = ((Status)Convert.ToInt16(txtStatus.Text)).ToString();
}
```

[VB] Handling the ItemDataBound Event

```
Enum Status
 Pending = 1
 Approved = 2
 Rejected = 3
 Complete = 4
End Enum
Private Sub statusDataTextBox_ItemDataBound(ByVal sender As Object, ByVal e As
System.EventArgs)
 Dim txtStatus As Telerik.Reporting.Processing.TextBox = DirectCast(sender,
```

```
Telerik.Reporting.Processing.TextBox)
 txtStatus.Text = (DirectCast(Convert.ToInt16(txtStatus.Text), Status)).ToString()
End Sub
```

Events and the Report Life Cycle

Events fire for report, sections and report items during the course of the report life cycle.

Report Definition

The Report Definition is created during the first stage of the life cycle. This is the actual .NET class that represents the report. It is always a subclass of ***Telerik.Reporting.Report*** and contains information about report items and their properties. Report items are represented by the private fields of the report class.

For example, if you add a TextBox to the Detail Section of the report in the designer, a private field of type TextBox is automatically added to the generated code-behind. This object will later serve as the definition for creating a concrete instance of the TextBox for each row in the data source.

Items defined in this phase of the life cycle, i.e. "definition items", are from the **Telerik.Reporting** namespace.

Report Processing

The second stage of the report life cycle involves combining the **report definition** with the **data source**. The processing engine:

- Performs all grouping, sorting and filtering calculations
- Iterates over all rows from the data source
- Creates the appropriate **processing items** based on the item definitions created earlier and the actual data.

Objects produced in this phase of the life cycle are called "processing items" and are from the **Telerik.Reporting.Processing** namespace.

Based on the original item definition (Telerik.Reporting.TextBox for example) and the actual data in the current data row a new item is created. A **processing item** (Telerik.Reporting.Processing.TextBox for example) bears all characteristics of its definition item, but is bound to the respective data field from the current data row. For example, the definition TextBox **Value** property may contain something like “=Fields.FirstName”, but the processing item **Text** property will be equal to “John”.

Data Binding During Processing

Just *before* the processing item is bound to data, the **ItemDataBinding** event of its definition item is raised. Then the processing item is bound to data and finally the **ItemDataBound** event is raised.

Suppose that we have a TextBox containing employee job positions. Assume that the TextBox **Value** property is “=Fields.JobPosition”, where "JobPosition" is a column from the data source containing values like “Team Leader”, “Senior Developer”, “Quality Assurance”, etc. For the sake of example, let's further suppose that we want to highlight the developers column text in a blue color.

First we need to attach to the TextBox ItemDataBound event. The ***sender*** parameter of the event handler method is in fact the **processing report item** and has already been data bound. Its **Text** property will hold the job position of the current employee. The event handler will be called for each data row, i.e. for each employee we have in the data source.

In this example, if Text contains "Developer", then the TextBox Style property is used to color the background a blue color.

[C#] Handling the ItemDataBound Event

```
private void textBox1_ItemDataBound(object sender, EventArgs e)
{
```

```
    Telerik.Reporting.Processing.TextBox txtPosition =
(Telerik.Reporting.Processing.TextBox)sender;
    if (txtPosition.Text.Contains("Developer"))
    {
        txtPosition.Style.BackgroundColor = Color.Blue;
    }
}
```

[VB] Handling the ItemDataBound Event

```
Private Sub textBox1_ItemDataBound(ByVal sender As Object, ByVal e As EventArgs)
 Dim txtPosition As Telerik.Reporting.Processing.TextBox = DirectCast(sender,
Telerik.Reporting.Processing.TextBox)
 If txtPosition.Text.Contains("Developer") Then
  txtPosition.Style.BackgroundColor = Color.Blue
 End If
End Sub
```

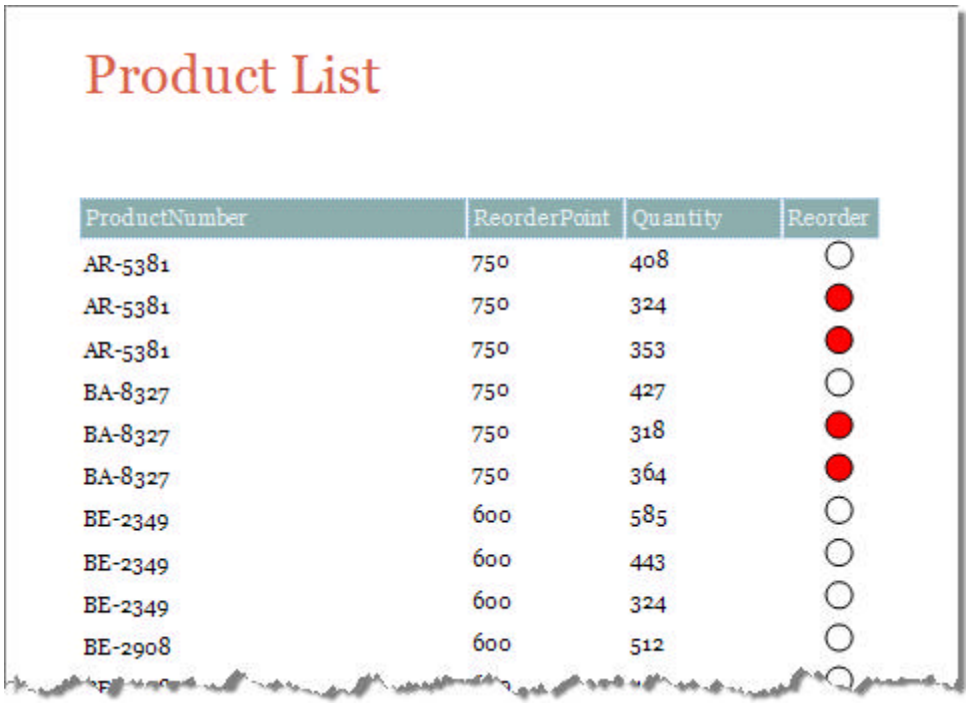
Report Rendering

After processing is over, the processed report is ready for rendering in one of the many available formats.

Lab: Creating an Event Handler for the Detail Section

The following walk-through demonstrates changing the appearance of one report item based on the data state of other bound columns. This product listing displays a red circle shape when the "Quantity" is less than half of the "ReorderPoint" value.

> The effect produced here could be implemented using Conditional Formatting and Expressions. The point here is to show the basic pattern that can be used for unusual issues where Conditional Formatting is not suited or does not provide sufficient control.



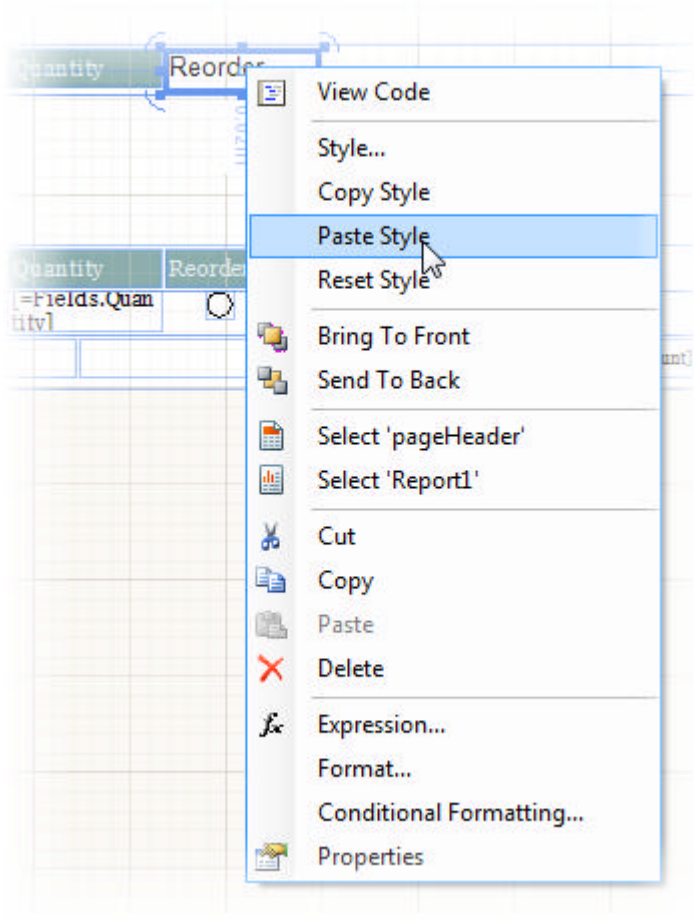
1. Create a new report class.

2. When the Report Wizard displays the Welcome to the Telerik Report Wizard page, click **Next**.
3. In the Report Choice page select **New Report** and click **Next**.
4. In the Choose Data Source page select **Create New Data Source**
5. In the Choose Data Source Type page click **Next**.
6. In the Choose Data Connection page click **New Connection**.
7. In the **Add Connection** dialog:
   1. Set the Data Source to be Microsoft SQL Server
   2. Set the server name to be <your server>\SQLEXPRESS
   3. Use Windows Authentication and select AdventureWorks as the database name.
   4. Click **OK** to close the Add Connection dialog.
8. In the Choose Data Connection page, select the database connection you just created and click **Next**.
9. In the Choose your Database Objects page, paste the query below and click **Next**:

   [SQL] Product Categories Query

   ```
   SELECT
   Production.Product.ProductNumber,
   Production.Product.ReorderPoint,
   Production.ProductInventory.Quantity
   FROM Production.Product
   INNER JOIN Production.ProductInventory
   ON Production.Product.ProductID = Production.ProductInventory.ProductID
   ```

10. On the Select Report Type page of the wizard, select **Standard** and click **Next**.
11. In the Design Data Layout page, populate the Detail section with "ProductNumber", "ReorderPoint" and "Quantity". Click the **Next** button.
12. In the Choose Report Layout page of the wizard, click the **Next** button.
13. In the Choose Report Style page of the wizard, select Civic and click **Next**.
14. In the Completing the Wizard page of the wizard, click the **Finish** button to close the wizard.
15. Add a **Shape** report item from the ToolBox. Place the shape in the Detail section to the right of the other columns. Set the **ShapeType** property to Ellipse.
16. Add TextBox report items from the ToolBox to the PageHeader and ReportHeader sections. Double-click each and enter "Reorder".
17. In the PageHeader, right-click the "Quantity" column TextBox and select "Copy Style" from the context menu. Right-click the "Reorder" TextBox and select "Paste Style" from the context menu. Now all the column headings should have the same style:



18. In the report designer, select the detail section with the mouse. In the Property Window, click the events button (⚡). Double click the **ItemDataboundEvent** to create an event handler.
19. Replace the event handler code with the code below.

[C#] Handling the Detail Section ItemDataBound Event

```
private void detail_ItemDataBound(object sender, System.EventArgs e)
{
   // Get the detail section object from sender
   Processing.DetailSection section = (Processing.DetailSection)sender;
   // From the section object get the DataRowView
   DataRowView dataRowView = (DataRowView)section.DataItem;
   // Also from the section object get the items in the report
   Processing.ReportItemBase[] items =
           (Processing.ReportItemBase[])section.Items.Find("shapeReorder", false);
   /// Get the specific report item you want to change,
   /// use the dataRowView to get the underlying data,
   /// then change the report item properties based on the data.
   if (items.Length > 0)
   {
       Processing.Shape shape = items[0] as Processing.Shape;
       int quantity = Convert.ToInt16( dataRowView["Quantity"]);
       int reorderPoint = Convert.ToInt16( dataRowView["ReorderPoint"]);
       bool timeToReorder = quantity < reorderPoint * .5;
       shape.Style.BackgroundColor = timeToReorder ? Color.Red : Color.White;
```

```
    }
}
```

[VB] Handling the Detail Section ItemDataBound Event

```
Private Sub detail_ItemDataBound(ByVal sender As Object, ByVal e As System.EventArgs)
 ' Get the detail section object from sender
 Dim section As Processing.DetailSection = DirectCast(sender, Processing.DetailSection)
 ' From the section object get the DataRowView
 Dim dataRowView As DataRowView = DirectCast(section.DataItem, DataRowView)
 ' Also from the section object get the items in the report
 Dim items As Processing.ReportItemBase() = DirectCast(section.Items.Find("shapeReorder",
False), Processing.ReportItemBase())
 ''' Get the specific report item you want to change,
 ''' use the dataRowView to get the underlying data,
 ''' then change the report item properties based on the data.
 If items.Length > 0 Then
  Dim shape As Processing.Shape = TryCast(items(0), Processing.Shape)
  Dim quantity As Integer = Convert.ToInt16(dataRowView("Quantity"))
  Dim reorderPoint As Integer = Convert.ToInt16(dataRowView("ReorderPoint"))
  Dim timeToReorder As Boolean = quantity < reorderPoint * 0.5
  shape.Style.BackgroundColor = IIf(timeToReorder,Color.Red,Color.White)
 End If
End Sub
```

This code gets a reference to the detail section, gets the section DetailItem property and casts it to be a DetailRowView. Once you have the DetailRowView you can access any bound column data. The detail section reference is also used to locate the Shape processing item from the section's Items collection. If the Shape item exists, the Shape processing item style is altered based on evaluating the column data.

> Remember that the reporting runtime object model is different from the design-time model. You should use the **Telerik.Reporting.Processing** namespace objects within event handlers.

Summary

In this section of the tutorial you became familiar with the basic report life cycle. Using your knowledge of the life cycle, you know when data bound fields can be used in contrast with global objects such as PageNumber or PageCount. You have also learned how to handle report, report item and section level events.

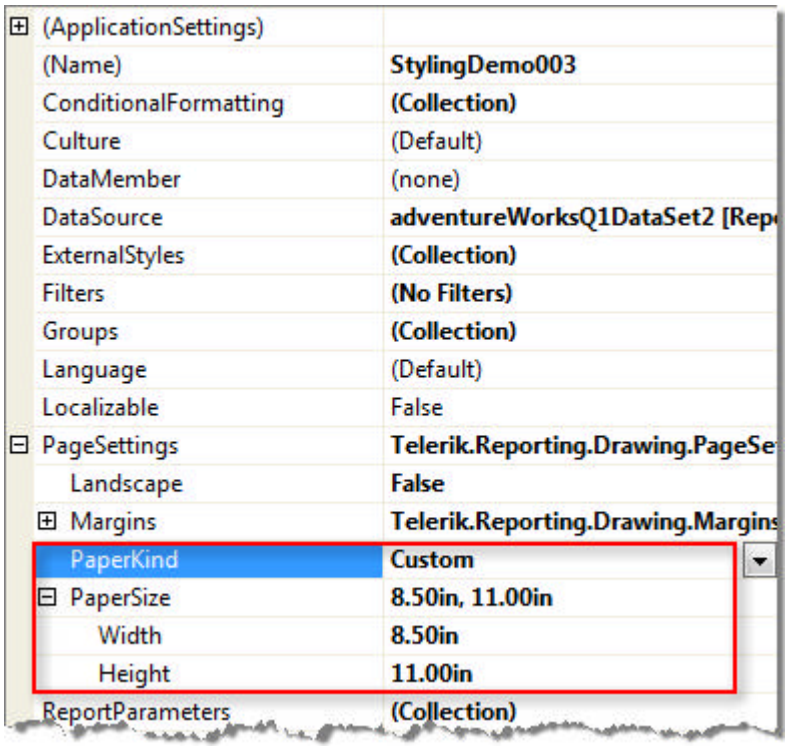
16 Paper Size

Objectives

This section will show you how to print sheets of labels from a datasource, how to print data that displays in multiple columns, and how to create reports that display in custom paper dimensions.

Paper Size Basics

Not all reports are 8.5 X 11 or legal size paper. Telerik Reports lets you choose from a number of formats or even create your own custom sizes for unusual cases. In the **PageSettings** property for the report, you can set the **PaperKind** to any of a large number of predefined settings, i.e. Letter, Legal, Statement, A3, etc. or you can set the **PaperKind** property to be **Custom**. That will allow you to edit the **PaperSize** property **Width** and **Height** to any custom dimensions that suit your purpose.

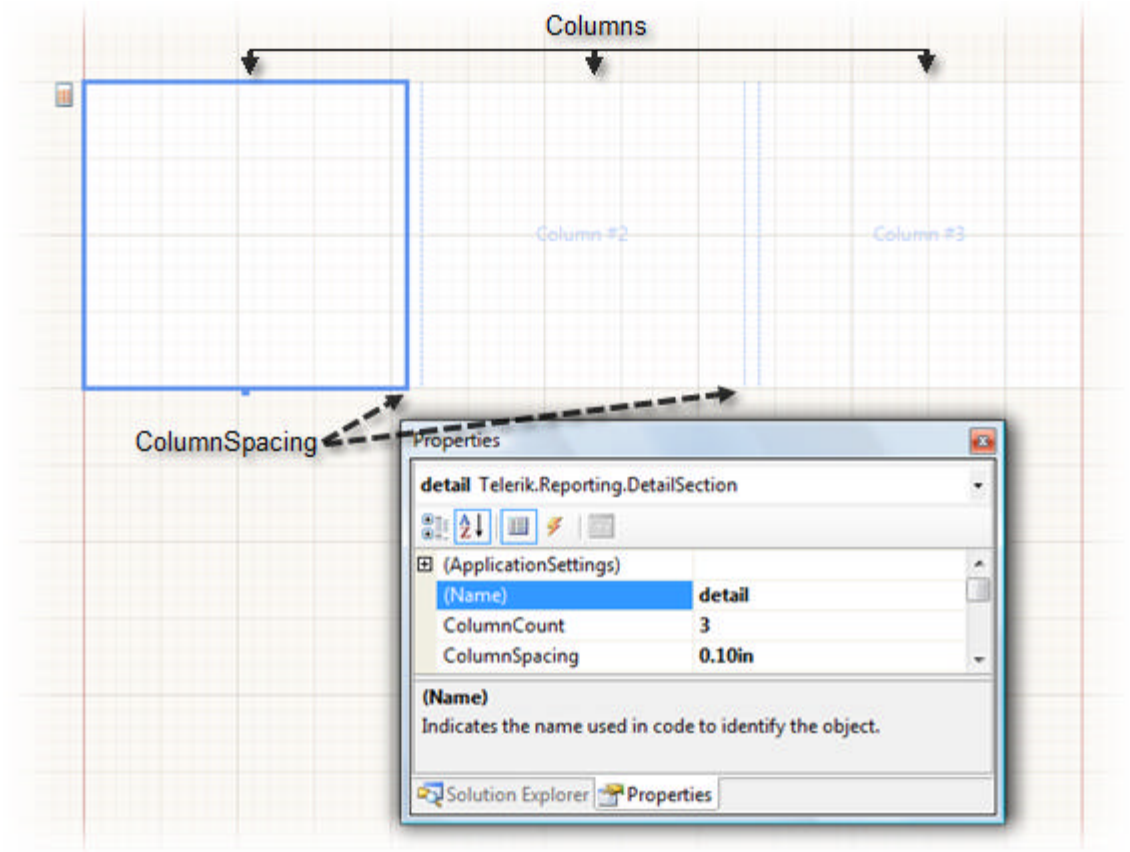


Multi Column Reports

By default, report data prints in a single column. By changing the Detail section **ColumnCount** and **ColumnSpacing** properties you can have the data automatically distributed over the new columns.

1. Create a new Report, but cancel the Report Wizard.
2. Remove the page header and footer.
3. Select the Detail section and in the Properties window, set the **ColumnCount** to "3" and **ColumnSpacing** to "0.10in". Also set the **Width** property to "2in". Notice that the design surface now allows you to edit in the first column and has place holder displays for the other columns.

> We reduce the width of the detail so that the detail column widths and spacing widths together fit within the page width. Page width is found in the Report PageSettings.PaperSize.

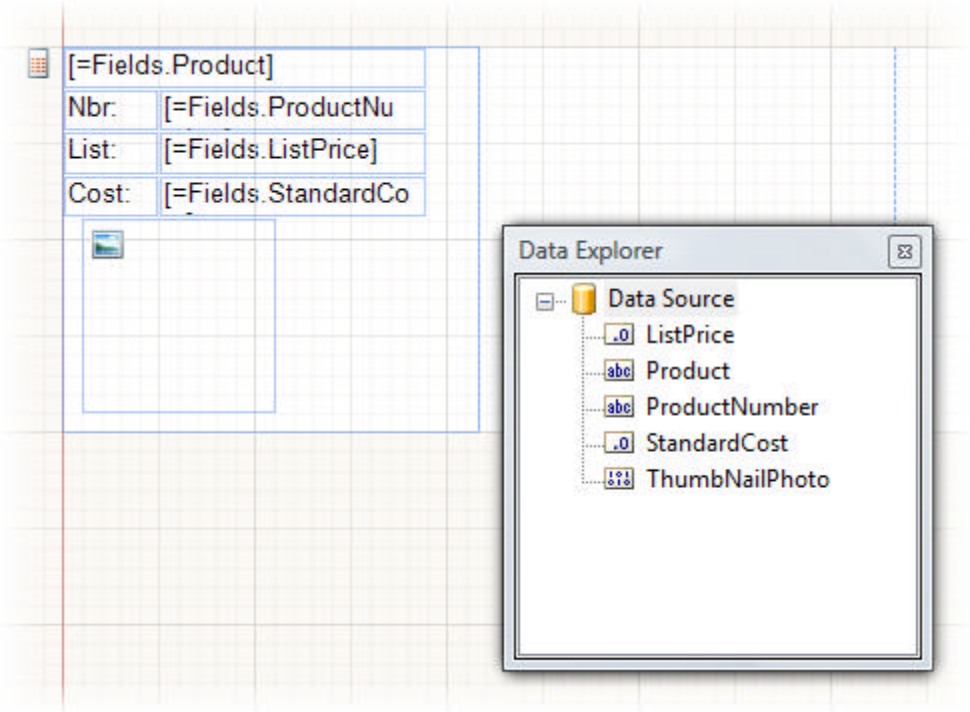


4. Create a new DataSource for the project using the SQL below:

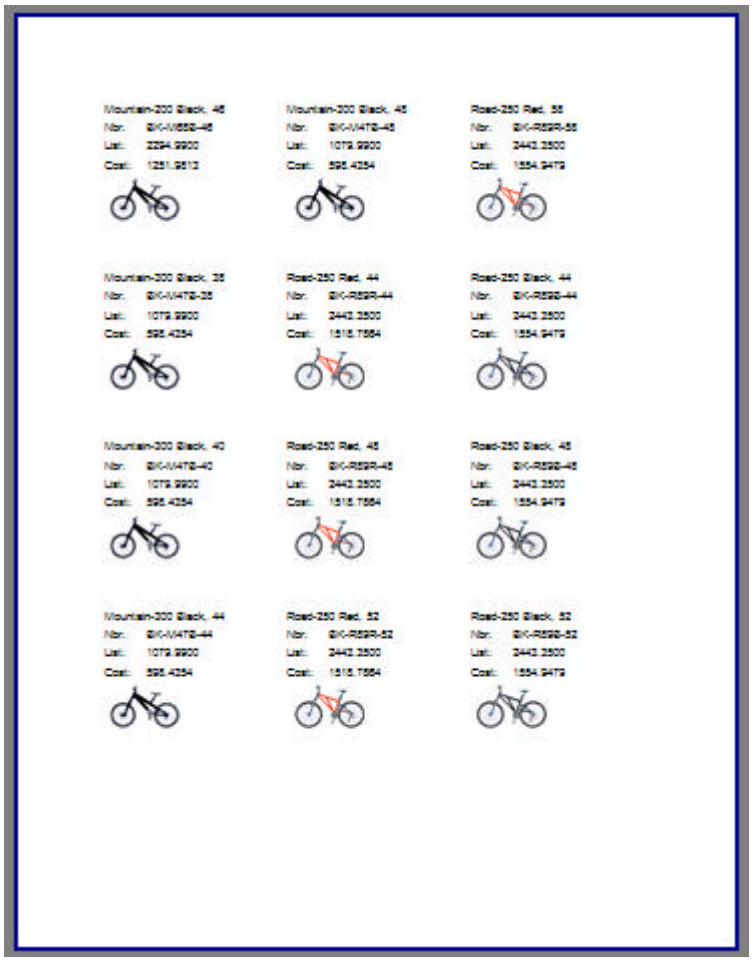
   [SQL] List Products

```
SELECT Production.Product.NAME AS Product,
Production.ProductPhoto.ThumbNailPhoto,
Production.Product.ProductNumber,Production.Product.ListPrice,
Production.Product.StandardCost
FROM Production.Product
INNER JOIN Production.ProductProductPhoto
ON Production.Product.ProductID = Production.ProductProductPhoto.ProductID
INNER JOIN Production.ProductPhoto
ON Production.ProductProductPhoto.ProductPhotoID = Production.ProductPhoto.ProductPhotoI
```

5. Using the Data Explorer, add the fields to the editable area of the Detail section. You may also want to add TextBox controls to act as labels. Place the "ThumbNailPhoto" at the bottom of the column:



6. Click the Preview tab to view the report.



Lab: Labels

Telerik Reports supports an enormous range of label types and size layouts. The Report Wizard includes a Labels option that helps guide you through the process of locating, choosing and configuring the right layout for your purposes.

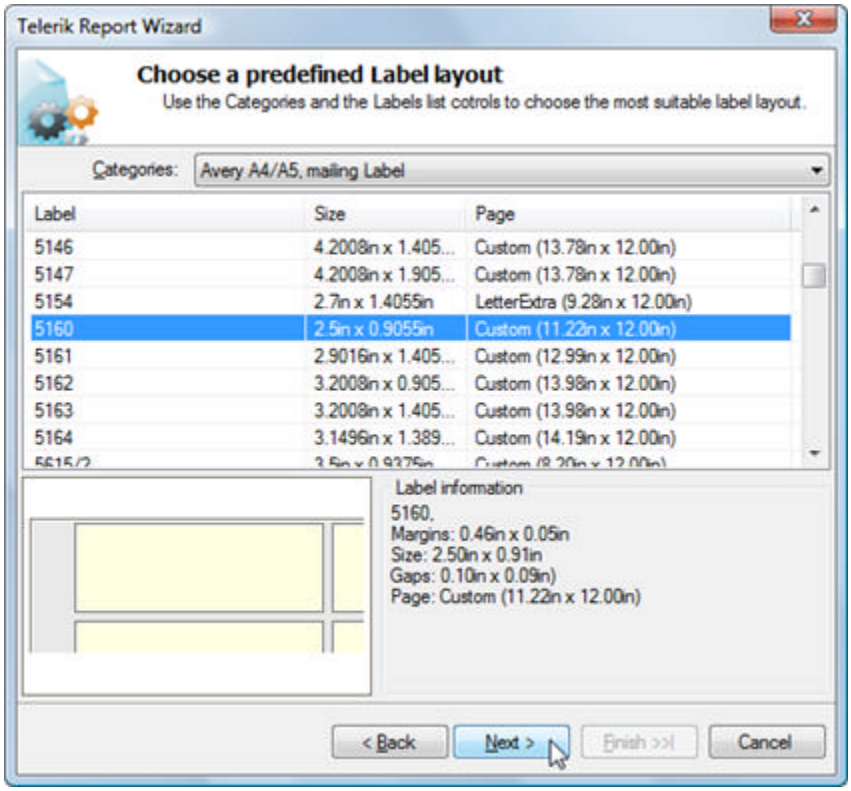
1. Using the Report Wizard, create a new Report based off the SQL below:

   [SQL] Address Listing

```
SELECT TOP (30)
Person.Contact.FirstName,
Person.Contact.LastName,
Person.Address.AddressLine1,
Person.Address.AddressLine2,
Person.Address.City,
Person.StateProvince.StateProvinceCode AS State,
Person.Address.PostalCode
FROM
Person.Address
INNER JOIN Sales.SalesOrderHeader
ON Person.Address.AddressID = Sales.SalesOrderHeader.ShipToAddressID
INNER JOIN
Person.Contact
ON Sales.SalesOrderHeader.ContactID = Person.Contact.ContactID
```

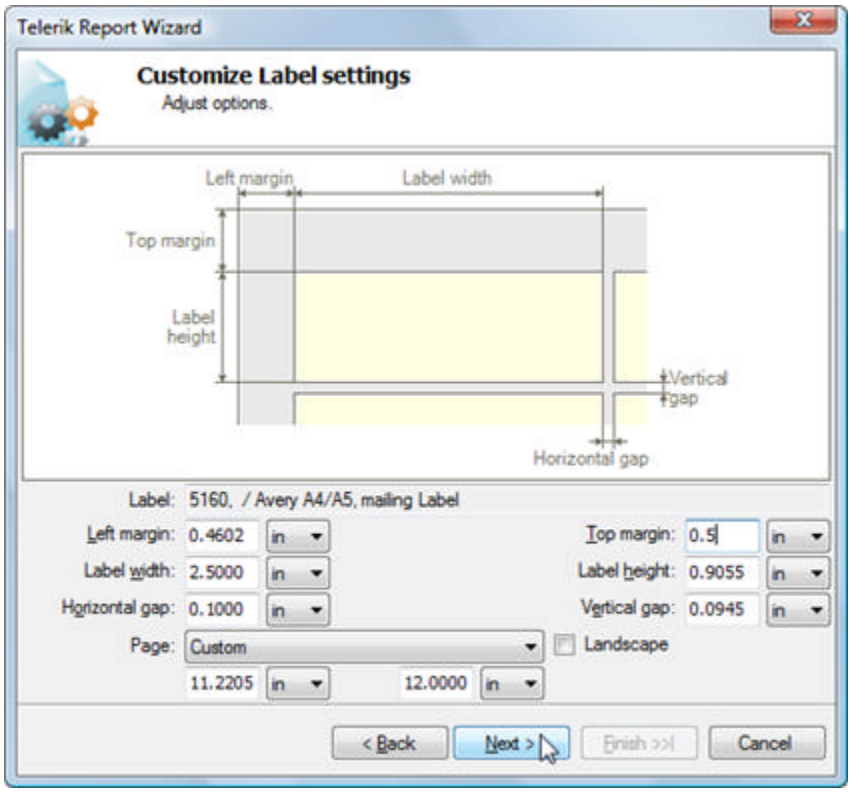
```
INNER JOIN Person.StateProvince
ON Person.Address.StateProvinceID = Person.StateProvince.StateProvinceID
```

2. When you get to the **Select a Report Type** page of the Report Wizard, select **Label.**
3. In the **Design Data Layout** page of the wizard, add all the fields to the **Bound Fields** list.
4. In the **Choose a Predefined Label Layout** page of the wizard, drop down the **Categories** list and select "Avery A4/A5 Mailing label". From the list of label layout types select the "5160" label.

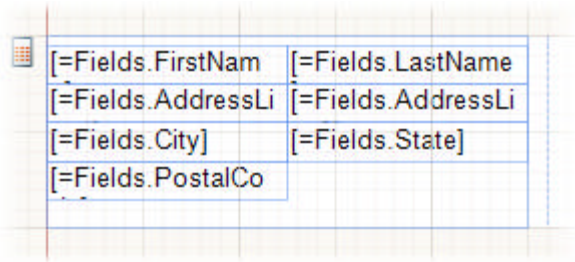


5. In the Customize Label Settings page, set the Top margin to ".5" and tab off the entry field to see the change in the preview area.

> Because of the WYSIWYG nature of Telerik Reporting, you can actually measure the sheet of paper to get the margins. For instance, if you measure a half an inch from the edge of the paper to the first label, you can enter that amount as the top margin here. You can use any of the measurement systems: mm, cm, in, px, pt, pc.

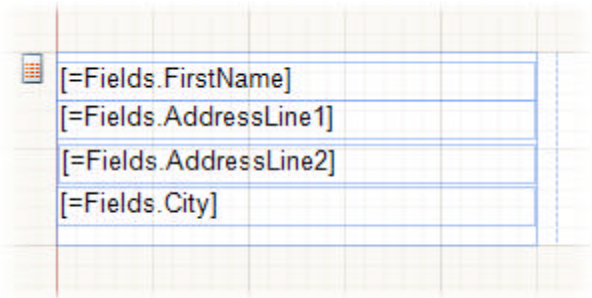


6. Finish the wizard and view the layout.



It needs a little help because the fields are all equal sized and not formatted appropriately for address labels.

7. Delete the TextBoxes "=Fields.LastName", "=Fields.State" and "=Fields.PostalCode".
8. Rearrange the TextBox controls so that each one appears on a line as shown in the screenshot below.



9. Set the **Value** property for the "=Fields.FirstName" TextBox to

=Fields.FirstName + " " + Fields.LastName

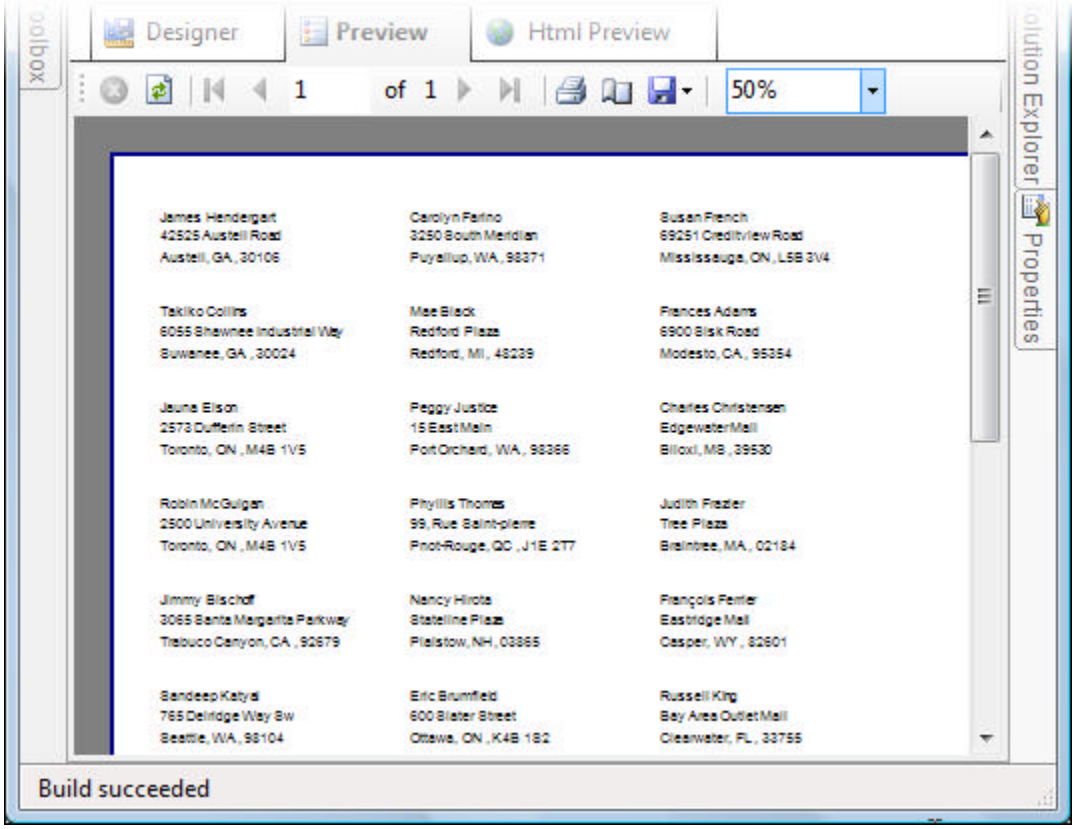
10. Set the **Value** property for the "=Fields.City" TextBox to

    =Fields.City + ", " + Fields.State + ", " + Fields.PostalCode

11. Click the **Preview** tab. Thats better, but there's a blank line when there's no data in an address line.

    Back in the designer, select both "AddressLine" TextBox items (Hold down the shift key and mouse click both items). In the properties window, set the **CanShrink** property to **True**.

12. Click the **Preview** tab. The labels now display in an appropriate arrangement.



> The real proof is in the printing. When you print on actual, not virtual label sheets, the Customize Label Settings page of the wizard is invaluable for adapting to your physical printing setup.

Summary

In this section you learned how to print labels from a database of addresses, how to display data in multiple columns and how to create reports in any size configuration.

17 Handling Report Output

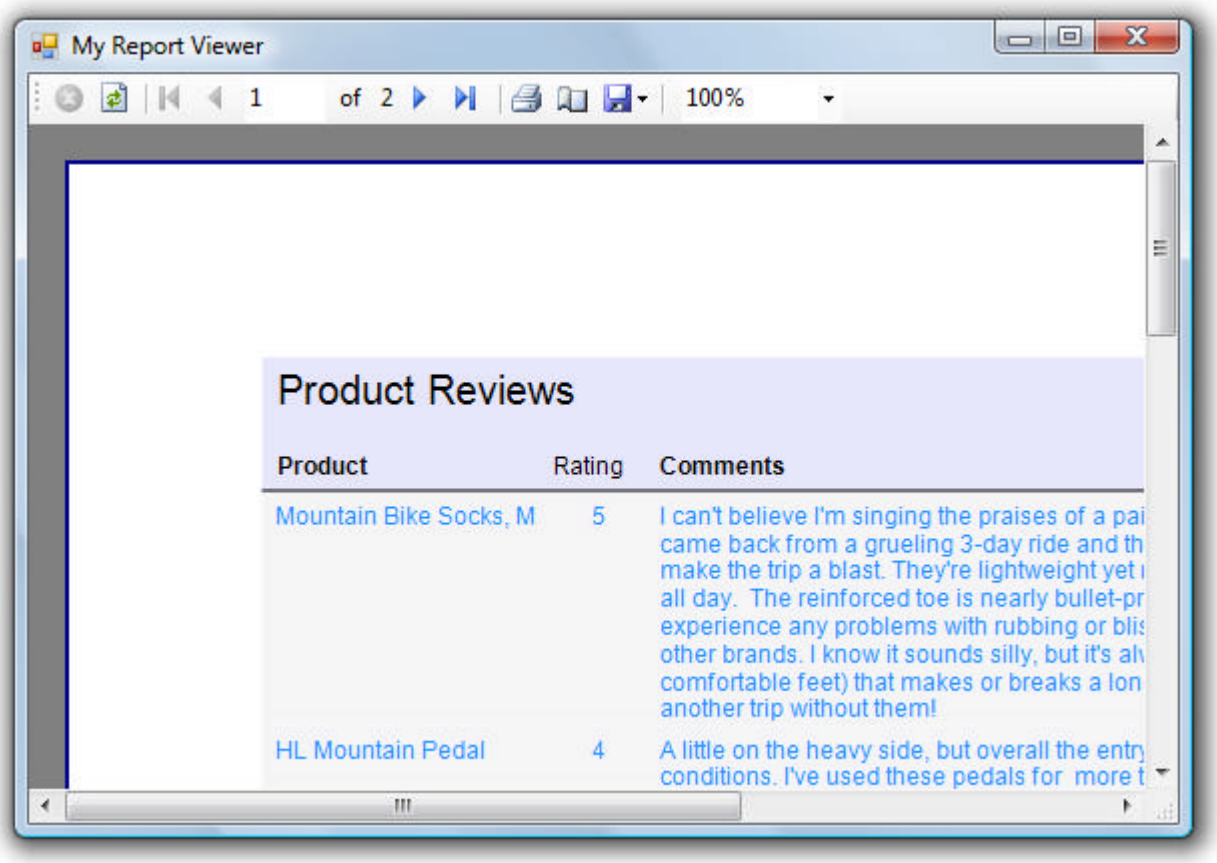
Objectives

In this section you will become familiar with two Telerik Reporting controls for viewing your reports within Windows and Web applications. You will also export your reports programmatically by rendering a report to a stream of bytes that can be written to various file formats on disk, including image and PDF.

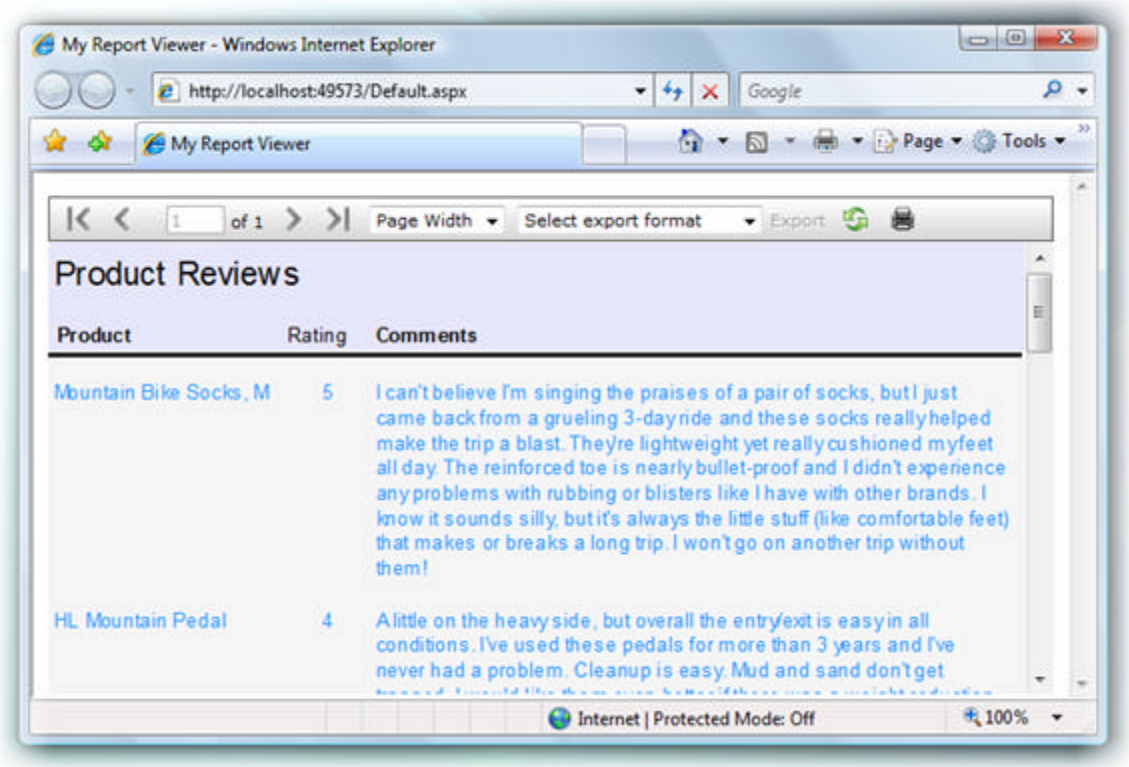
Report Output Basics

Telerik Reporting comes with two controls for viewing your reports within Windows and Web applications. You can also export your reports programmatically by rendering your report to a stream of bytes that can be written to various file formats on disk, including image and PDF.

The ReportViewer for Windows Applications



The ReportViewer for Web Applications



The toolbar for both viewers allows the user to interact with the currently-loaded report:

- Stop loading (useful when a large report is taking a long time to load)
- Refresh
- Go to first page
- Go to preview page
- Go to a specific page
- Total number of pages
- Go to next page
- Go to last page
- Print
- Export: Reports can be exported to Web Archive (MSHTML), TIFF, PDF, RTF, Excel and comma delimited files. Select a format, then click the **Export** button.
- Zoom level

Windows Report Viewer

The **ReportViewer** control used in Windows applications has properties to assign the report content and to control the toolbar display.

- Set the **Report** property of the viewer using the drop down list in the **Properties** Window. Be sure to reference the assembly containing the report so that the report shows up in the drop down. Also be sure to build the application so that reports show up in the drop down list.
- To size the report contents to the **ReportViewer** window use the **ZoomMode** property. **ZoomMode** can be

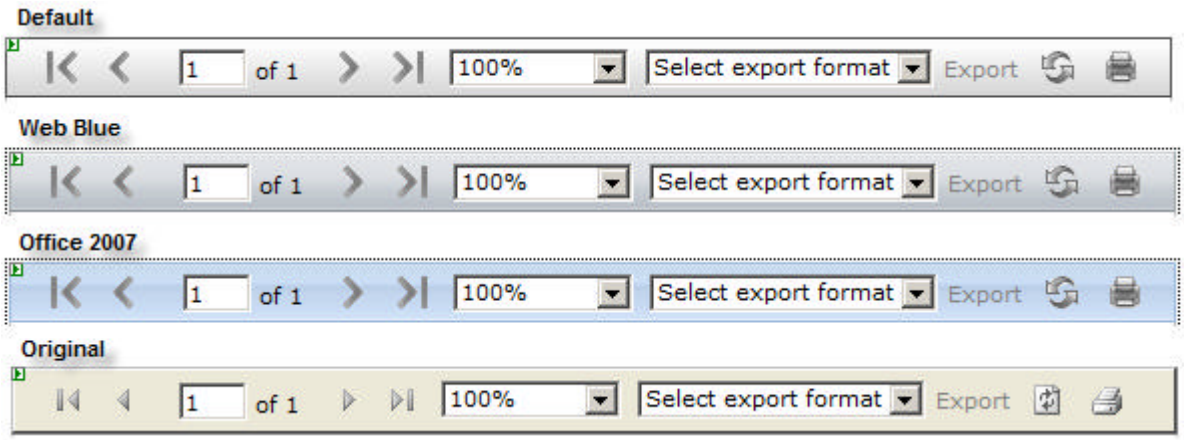
set to **FullPage**, **PageWidth** or **Percent**. If set to **ZoomPercent**, also set the **ZoomPercent** property to a value greater than zero.

- To tailor the controls available in the **ReportViewer**, set the boolean properties **ShowExportButton**, **ShowPageNavigationControls**, **ShowPageSetup**, **ShowParametersButton**, **ShowPrintButton**, **ShowProgress**, **ShowRefreshButton**, **ShowZoom**.
- The **CurrentPage** property defaults to zero but can be used to start at another page in the report.

Web Report Viewer

The properties that the Web **ReportViewer** uses are similar to the Windows Application version, with a couple of additions:

- **ProgressText:** The text that displays in the viewer while the report is being generated. The default is "Generating report...".
- **Skin**: Select from one of the predefined skins (see screenshot below).



Exporting Reports Programmatically

To export a report, you can use the **Render()** method of the **ReportProcessor** object. This method converts the contents of the report to a byte array in the specified format. Your project must contain references to **Telerik.Reporting**, **Telerik.Reporting.Processing**, and the assembly containing your report class.

Exporting to PDF

For example, you can export directly to a PDF file and have it created right on local disk:

1. Open one of the previous solutions that contains a defined report, such as the "Adding Style Rules" project.
2. In the Solution Explorer, right-click the solution and select **Add | New Project** from the context menu.
3. Select Windows Application, provide a unique project name and click OK.
4. Also in the Solution Explorer, right click the **References** node and **Add Reference...**
5. Add a reference to **Telerik.Reporting** and **Telerik.Reporting.Processing**. You can use the **Browse** tab of the **Add Assembly** dialog to find these assemblies in your Telerik Reporting installation directory under the \Bin folder.
6. Also add a reference to the assembly that contains your report class. Use the **Projects** tab to list the other assemblies in your solution.
7. Add a button to the default form.
8. Double-click the button to create a Click event handler. Add the code below to the event handler.

Note: Change the Reference to "MyReport" to the class name of the report class you referenced.

The code creates an instance of the report and calls ReportProcessor.Render(), passing the report object as a parameter and receiving a raw array of bytes in return. The bytes are written to a FileStream object.

[C#] Exporting to PDF

```
private void button1_Click(object sender, EventArgs e)
{
   // create an instance of the report
   MyReport myReport = new MyReport();
   // initialize parameters to be used in Render()
   string mimType = string.Empty;
   string extension = string.Empty;
   Encoding encoding = null;
   // call Render() and retrieve raw array of bytes
   byte[] buffer = Telerik.Reporting.Processing.ReportProcessor.Render(
       "PDF", myReport, null, out mimType, out extension, out encoding);
   // create a new file on disk and write the byte array to the file
   FileStream fs = new FileStream("c:\\report1.pdf", FileMode.Create);
   fs.Write(buffer, 0, buffer.Length);
   fs.Flush();
   fs.Close();
}
```

[VB] Exporting to PDF

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
 ' create an instance of the report
 Dim myReport As New MyReport()
 ' initialize parameters to be used in Render()
 Dim mimType As String = String.Empty
 Dim extension As String = String.Empty
 Dim encoding As Encoding = Nothing
 ' call Render() and retrieve raw array of bytes
 Dim buffer As Byte() = Telerik.Reporting.Processing.ReportProcessor.Render("PDF",
myReport, Nothing, mimType, extension, encoding)
 ' create a new file on disk and write the byte array to the file
 Dim fs As New FileStream("c:\report1.pdf", FileMode.Create)
 fs.Write(buffer, 0, buffer.Length)
 fs.Flush()
 fs.Close()
End Sub
```

9. Add references to System.IO and your report class assembly in the "uses" (C#) or "Imports" (VB) section of code.
10. In the solution explorer, right-click the project and select **Set as Startup Project** from the context menu.
11. Press **F5** to run the application. Click the button and examine the report output to your "C:" drive.

Exporting to Image

Now try swapping out the button click event handler code with the code below. You will need to add a **System.Collections** reference to support the Hashtable. Also, change the reference to MyReport to use the name of the report class you referenced.

This example has a similar setup to the previous example, but the returned array of bytes is handled differently. The bytes are plugged into a MemoryStream constructor and the stream is used as a parameter to

Image.FromStream() to create an Image object, and finally the Image.Save() method is called to persist the image file to disk.

[C#] Exporting to Image File

```
private void button1_Click(object sender, EventArgs e)
{
   // create an instance of the report
   MyReport myReport = new MyReport();
   // initialize variables to be used as render parameters
   string mimType = string.Empty;
   string extension = string.Empty;
   Encoding encoding = null;
   // create a hashtable to contain the output format as expected by Render()
   Hashtable deviceInfo = new Hashtable();
   deviceInfo["OutputFormat"] = "JPEG";
   // make the Render call and retrieve a raw array of rendered bytes
   byte[] buffer = Telerik.Reporting.Processing.ReportProcessor.Render(
       "IMAGE", myReport, deviceInfo, out mimType, out extension, out encoding);
   // plug the byte array into a stream, and from the stream into an Image
   using (MemoryStream stream = new MemoryStream(buffer))
   {
       using (Image img = Image.FromStream(stream))
       {
           // persist the image to disk
           img.Save(@"c:\" + myReport.GetType().Name + "." + extension,
             System.Drawing.Imaging.ImageFormat.Jpeg);
       }
   }
}
```

[VB] Exporting to Image File

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
 ' create an instance of the report
 Dim myReport As New MyReport()
 ' initialize variables to be used as render parameters
 Dim mimType As String = String.Empty
 Dim extension As String = String.Empty
 Dim encoding As Encoding = Nothing
 ' create a hashtable to contain the output format as expected by Render()
 Dim deviceInfo As New Hashtable()
 deviceInfo("OutputFormat") = "JPEG"
 ' make the Render call and retrieve a raw array of rendered bytes
 Dim buffer As Byte() = Telerik.Reporting.Processing.ReportProcessor.Render("IMAGE",
myReport, deviceInfo, mimType, extension, encoding)
 ' plug the byte array into a stream, and from the stream into an Image
 Using stream As New MemoryStream(buffer)
  Using img As Image = Image.FromStream(stream)
   ' persist the image to disk
   img.Save("c:\" + myReport.[GetType]().Name + "." + extension,
System.Drawing.Imaging.ImageFormat.Jpeg)
  End Using
 End Using
End Sub
```

Emailing a Report

This last example requires an SMTP server configured and tested. The example is intended as a pointer on how to get the rendered report to an email attachment -- not a primer on SMTP.

Important: You should configure and test programmatically sending SMTP messages with attachments before adding the last step where you use the report bytes as the attachment.

The example renders a report to a stream of bytes, similar to the examples above and uses the stream to populate an email attachment.

[C#] Creating a Report Attachment

```
private void button1_Click_1(object sender, EventArgs e)
{
   MailReport(new MyReport(), "me@myinc.com", "someoneelse@somewhereelse.com",
       "Product Reviews", "Here is the product review report");
}
void MailReport(Telerik.Reporting.Report report, string from, string to, string
subject, string body)
{
   // initialize parameters for use in the call to Render()
   string mimeType;
   string extension;
   Encoding encoding;
   // call Render() and retrieve the rendered array of bytes
   byte[] reportBytes =
       ReportProcessor.Render("PDF", report, null, out mimeType, out extension, out
encoding);
   // place the bytes into a memory stream
   MemoryStream ms = new MemoryStream(reportBytes);
   ms.Position = 0;
   // create an email attachment that consumes the memory stream and send it
   Attachment attachment = new Attachment(ms, report.Name + "." + extension);
   MailMessage msg = new MailMessage(from, to, subject, body);
   msg.Attachments.Add(attachment);
   SmtpClient client = new SmtpClient("hostname");
   client.Send(msg);
}
```

[VB] Creating a Report Attachment

```
Private Sub button1_Click_1(ByVal sender As Object, ByVal e As EventArgs)
 MailReport(New MyReport(), "me@myinc.com", "someoneelse@somewhereelse.com", "Product
Reviews", "Here is the product review report")
End Sub
Sub MailReport(ByVal report As Telerik.Reporting.Report, ByVal from As String, ByVal
[to] As String, ByVal subject As String, ByVal body As String)
 ' initialize parameters for use in the call to Render()
 Dim mimeType As String
 Dim extension As String
 Dim encoding As Encoding
 ' call Render() and retrieve the rendered array of bytes
 Dim reportBytes As Byte() = ReportProcessor.Render("PDF", report, Nothing, mimeType,
extension, encoding)
 ' place the bytes into a memory stream
 Dim ms As New MemoryStream(reportBytes)
 ms.Position = 0
 ' create an email attachment that consumes the memory stream and send it
```

```
    Dim attachment As New Attachment(ms, report.Name + "." + extension)
    Dim msg As New MailMessage(from, [to], subject, body)
    msg.Attachments.Add(attachment)
    Dim client As New SmtpClient("hostname")
    client.Send(msg)
   End Sub
```

Display a PDF Directly in the Browser

If you want to skip the export step and display a rendered PDF directly in a browser without saving it to disk first, the technique below writes the byte array returned by Render() directly to the Response stream:

1. Add a Web Application to the previous solution used in "Exporting".
2. Add a standard ASP.NET button to the default page.
3. Add a reference to **Telerik.Reporting** and **Telerik.Reporting.Processing**. You can use the **Browse** tab of the **Add Assembly** dialog to find these assemblies in your Telerik Reporting installation directory under the \Bin folder.
4. Double-click the button to create a Click event handler.
5. Replace the Click event handler with a call to DisplayPDF(), and the DisplayPDF() method itself using the code example below.

[C#] Write PDF Bytes to Response Stream

```
void DisplayPDF(string reportName, Telerik.Reporting.Report reportToExport)
{
   // initialize parameter variables
   string mimeType = string.Empty;
   string ext = string.Empty;
   System.Text.Encoding encoding = System.Text.Encoding.Default;
   // call Render() and retrieve array of bytes
   byte[] reportBytes =
      Telerik.Reporting.Processing.ReportProcessor.Render("PDF", reportToExport, null,
       out mimeType, out ext, out encoding);

   // inject the bytes to the Response stream
   string fileName = reportName + ".pdf";
   Response.Clear();
   Response.ContentType = mimeType;
   Response.Cache.SetCacheability(HttpCacheability.Private);
   Response.Expires = -1;
   Response.Buffer = false;
   Response.AddHeader("Content-Disposition",
       string.Format("{0};FileName=\"{1}\"", "attachment", fileName));
   Response.OutputStream.Write(reportBytes, 0, reportBytes.Length);
   Response.End();
}

protected void Button1_Click(object sender, EventArgs e)
{
   DisplayPDF("Product Reviews", new MyReport());
}
```

[VB] Write PDF Bytes to Response Stream

```
Sub DisplayPDF(ByVal reportName As String, ByVal reportToExport As
Telerik.Reporting.Report)
 ' initialize parameter variables
 Dim mimeType As String = String.Empty
 Dim ext As String = String.Empty
 Dim encoding As System.Text.Encoding = System.Text.Encoding.[Default]
 ' call Render() and retrieve array of bytes
 Dim reportBytes As Byte() = Telerik.Reporting.Processing.ReportProcessor.Render("PDF",
reportToExport, Nothing, mimeType, ext, encoding)

 ' inject the bytes to the Response stream
 Dim fileName As String = reportName + ".pdf"
 Response.Clear()
 Response.ContentType = mimeType
 Response.Cache.SetCacheability(HttpCacheability.[Private])
 Response.Expires = -1
 Response.Buffer = False
 Response.AddHeader("Content-Disposition", String.Format("{0};FileName=""{1}""",
"attachment", fileName))
 Response.OutputStream.Write(reportBytes, 0, reportBytes.Length)
 Response.[End]()
End Sub

Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
 DisplayPDF("Product Reviews", New MyReport())
End Sub
```

Deploying Reports to a Server

Deploying a report viewing application along with the reports is virtually the same process for Windows and Web Applications (except where noted). The basic steps are:

- Build the application.
- Set the Telerik Reporting assembly references Copy Local flag to True.
- Copy the contents of the bin directory to the server.
- Run the application.

Not covered here, but important considerations when deploying reports in a production setting are...

- Database client configuration and connectivity
- Permissions on the deployment server. The permissions and authentication for the deployment server will vary widely according to operating system, configuration and the type of application deployed (web vs. Windows).

The example below takes a bare-bones report with no database connectivity or other dependencies and displays it in a simple Windows viewer application. This example requires a separate server machine to copy the application files to. Deployment does **NOT** require:

- Any of the design-time assemblies including Telerik.Reporting.Design.dll, Telerik.Reporting.VsPackage.dll, Telerik.ReportViewer.Design.dll
- Assemblies involved in conversion from other reporting platforms. These assemblies are also only used at design time and may include Interop.CRAXDRT.dll, Telerik.ReportConverter.ActiveReports3.dll, Telerik.ReportConverter.CrystalReports11.dll, Telerik.ReportConverter.XtraReports7.dll.

> You may want to use the example here to test report deployment on your server, separate from any database issues as a diagnostic measure. The more minimal the report and viewing application, the easier it will be to debug more complex issues later.

1. In Visual Studio create a new solution with a new **Class Library** project.
2. Add a **Telerik Report** to the project, but cancel the Report Wizard.
3. Remove the page header and footer sections, leaving only the detail section.
4. Add a TextBox to the detail section. Double-click the TextBox and enter a "Hello World" sentence of your choice.
5. Add a **Windows Application** to the solution.
6. Add a reference to the report class library (you can right-click the References node in the Solution Explorer, select **Add Reference...** from the context menu and select the report project assembly from the **Projects** tab).
7. Drop a ReportViewer on the form.
8. In the Properties Window, locate the **Report** property for the ReportViewer and select the "Hello World" report from the drop down list.
9. In the Solution Explorer, open the References node.
10. Make sure the Telerik Reporting assemblies are included in the deployment:
    1. Right-click the **Telerik.Reporting** assembly. Select **Properties** from the context menu. Set the **Copy Local** property to **True**.
    2. Right-click the **Telerik.Reporting.Interfaces** assembly. Select **Properties** from the context menu. Set the **Copy Local** property to **True**.
    3. Right-click the **Telerik.Reporting.Processing** assembly. Select **Properties** from the context menu. Set the **Copy Local** property to **True**.
    4. Right-click the **Telerik.Reporting.Winforms** assembly. Select **Properties** from the context menu. Set the **Copy Local** property to **True**. Note: if this is a web site or a web application, the namespace will be **Telerik.Reporting.Webforms**.

    > The reason we are making these settings is to be sure the assemblies physically exist in the bin folder of the deployed application. During the installation of Telerik Reporting, Telerik assemblies are added to the Global Assembly Cache (GAC). When deploying an application using the output of Visual Studio, the assemblies from the GAC are not copied automatically.

11. Build the application.
12. Copy the contents of the Windows report viewer application to the server. The contents of the bin directory may look something like the screenshot below:

Name	Date modified	Type	Size
HelloWorldReport.dll	5/7/2008 11:47 AM	Application Extens...	16 KB
HelloWorldReport.pdb	5/7/2008 11:47 AM	PDB File	12 KB
ReportViewer.exe	5/7/2008 11:47 AM	Application	20 KB
ReportViewer.pdb	5/7/2008 11:47 AM	PDB File	20 KB
ReportViewer.vshost.exe	5/7/2008 11:45 AM	Application	6 KB
Telerik.Reporting.dll	4/16/2008 3:43 PM	Application Extens...	1,404 KB
Telerik.Reporting.Interfaces.dll	4/16/2008 3:43 PM	Application Extens...	19 KB
Telerik.Reporting.Interfaces.xml	4/16/2008 3:43 PM	XML Document	2 KB
Telerik.Reporting.Processing.dll	4/16/2008 3:43 PM	Application Extens...	821 KB
Telerik.Reporting.Processing.xml	4/16/2008 3:43 PM	XML Document	30 KB
Telerik.Reporting.xml	4/16/2008 3:43 PM	XML Document	328 KB
Telerik.ReportViewer.WinForms.dll	4/16/2008 3:43 PM	Application Extens...	108 KB
Telerik.ReportViewer.WinForms.xml	4/16/2008 3:43 PM	XML Document	8 KB

13. Test run the application from the server.

For more detail on deploying Telerik Reporting on a server
see **http://www.telerik.com/help/reporting/install_DeployingOnAServer.html**.

Summary

In this section you worked with the Windows and Web report viewer controls. You also programmatically rendered your report as an image and PDF. Finally, you learned how to render your PDF directly to a browser.

Telerik Reporting