

# SharpShooter Collection Documentation

---

Last modified on: September 14, 2011



## Table of Content

SharpShooter Reports .....	8
SharpShooter Reports Getting Started .....	8
Main Features.....	8
System Requirements .....	8
Installation .....	9
Editions .....	11
Deploy .....	11
Getting Started.....	12
Basic Information.....	22
Basic Terms .....	22
Main Components Used to Create a Report .....	23
Using Data Sources.....	23
Using Scripts.....	24
Binding Elements to Data.....	24
Document Generation.....	24
Visualization and Editing Final Documents.....	25
Export, Print and Saving .....	25
Report Designer Interface .....	26
Tabs.....	26
Contextual Tabs.....	34
Application Menu .....	44
Binding Bar .....	45
Quick Access Toolbar.....	46
Context Menu .....	48
Docking .....	53
Editors .....	56
Wizards .....	67
Status Bar .....	74



Hotkeys .....	75
Template Objects .....	77
Classification .....	77
Bands.....	77
Visual Elements .....	79
User Guide .....	81
Template .....	81
Page .....	84
Objects' Controls .....	89
Objects' Properties .....	90
Text Output .....	111
List .....	117
Grouping .....	126
Horizontal List and Table .....	130
SideBySide Bands.....	132
Pivot Table.....	133
Image .....	140
Using Shapes .....	144
Barcodes .....	146
Parameters .....	148
Hyperlinks and Report Navigation .....	149
Using subreports .....	153
Zip code .....	159
Master-Report .....	160
Creation of Reports Using Wizard .....	161
Using Product Features .....	172
Scripts.....	172
Data Sources .....	182
Export .....	193
Projects .....	199



Using Advanced Product Features.....	202
Custom Report Designer .....	202
Custom Report Viewer .....	203
Custom Wizard .....	204
Custom Report Element .....	204
Custom Export Filter .....	205
Custom Template Storage.....	205
Live Report .....	205
Managing Report Generation .....	206
Appendix 1 Document Object Model .....	208
SharpShooter Reports.Silverlight.....	211
Getting Started .....	211
Introduction .....	211
Product Prerequisite .....	211
Creating Sample Application.....	211
Conclusion .....	232
SharpShooter Gauges .....	233
Destination and Basic Features .....	233
SharpShooter Gauges Concepts .....	233
Getting Started .....	234
Elements Description .....	238
The General Instrument Model .....	238
Elements Hierarchy .....	239
Description of General Properties .....	245
The Description All Elements and Their Properties .....	246
Expressions .....	260
Destination and General Principles of the Expressions Use .....	260
Description of Expression Language Syntax and Semantics .....	260
Use and Capabilities of the Instrument Designer .....	266
Work in Windows Forms Applications .....	273





Components Used in Windows Forms Applications.....	273
The Widget Component Use and Capabilities. ....	273
The IndicatorWidget Component Use and Capabilities.....	275
Work in Web Forms Applications .....	279
Components Used in Web Applications. ....	279
The WidgetProducer and WidgetHolder Components Use and Capabilities .....	279
The Technology of the Instrument Design. ....	283
Designing the Simple Instrument with a Slider .....	283
Creation of Instruments Using Complex Expressions .....	286
Working With the OPCChannel Component .....	291
General Information .....	291
OPCConnection.....	291
OPCChannel .....	291
Step-by-step Example of Application Creation .....	292
Advanced DataChannel Design.....	293
Capabilities .....	293
Design.....	293
SharpShooter OLAP .....	294
Destination and Basic Features .....	294
Getting Started .....	294
Basic Information.....	300
Concept.....	300
Data Sources .....	300
Working with the DataCubeGrid Component (Pivot Table Presentation, Manipulation) .....	301
Expressions .....	309
Destination and General Principles of Expression Use .....	309
Description of the Expression Language Syntax .....	309
Work in Windows Forms Applications .....	315
Components Used in Windows Forms Applications.....	315
The Use and Capabilities of the DataCube .....	315



The Use and Capabilities of the DataCubeGrid .....	317
Working with Groups .....	323
The Work with Large Body of Data .....	331
Custom Cell Styles .....	332
Custom Drawing .....	333
The Use and Capabilities of DataCubePrintDocument .....	334
The Interaction of the SharpShooter OLAP with Third Party Components .....	335
The Interaction with SharpShooter OLAP by Means of the DataCubeView Component .....	337
SharpShooter Charts.....	339
Destination and Basic Features .....	339
General Principles of the SharpShooter Charts .....	339
Creation of a Simple Chart with the Use of Wizard .....	339
Creation of a Simple Chart without Using Wizard.....	349
Chart Model.....	359
Description of Common Properties .....	359
ChartArea .....	360
Chart Series.....	360
Series and Assignment of a Data Source.....	370
Data Binding .....	370
Axis and Axis Elements.....	371
Additional Chart Elements.....	375
Additional Features .....	380
The Chart Viewer .....	382
The Designer .....	384
Using DashboardViewer .....	389
Joint use of SharpShooter Charts and SharpShooter OLAP.....	391
Work in Web Forms Applications .....	396
Using charts in Web forms applications.....	396
Example of using charts.....	396
Using dashboard in Web forms application .....	398



Appendix 1 Working with the Expression Editor .....401



## SharpShooter Reports

### SharpShooter Reports Getting Started

SharpShooter Reports is a flexible .Net report generator that allows user to create reports of any complexity based on unlimited number of sources of different types and to export ready documents to different file formats. Product components can be used in Windows forms, ASP.NET pages and Silverlight apps. The package includes Designer that allows editing ready report templates and documents. The report can contain different elements. Using a built-in component one can dynamically analyze data with the help of Pivot tables. Data used in the report can be presented as charts and diagrams.

### Main Features

- .Net 4.0 compatibility.
- Completely controlled code.
- Compatibility with the popular integrated .NET development environments.
- WinForms, ASP.NET WebForms and Silverlight support.
- ADO.NET compatibility.
- Support any .NET objects as data source.
- Report generation wizard.
- Using any .NET language for expression and script creation.
- Data binding model.
- Easy processing of report transfer via Internet due to XML-based report files.
- Report viewer components for WinForms, ASP.NET WebForms and Silverlight.
- End user report designer.
- Unlimited data sources for a single report.
- Unlimited number of embedded and/or serial bands in the report.
- Built-in component for Pivot table building.
- Parameterized report creation and usage.
- Parameterized sub-reports creation and usage.
- SideBySide is advanced concept of a parallel report generation.
- Unified reports can be generated in a short period of time.
- Generated reports are stored as an object graph.
- Complete document object model is easy to use and can be used both with report templates and ready documents.
- Bound and unbound reports.
- Export to the most widespread file formats is supported.
- Manual section output control (optional).
- Crystal Report import option.
- Zip- and barcodes generation supported.
- Built-in wide range of visual elements.
- Opportunity to use GDI+ advances.
- Unlimited number of pages with different size and layout can be included in a report.
- Usage of headers and footers increases usability.
- Water marks can be laid on the background.
- Styles available.
- Several measuring systems supported.
- Detailed documentation and extensive base of examples available.
- Easy deployment into your applications.
- Distribution of SharpShooter Reports inside your application is free and does not require additional royalties.
- Competent and up-to-date technical support in Russian and English is provided.

### System Requirements

In order to create applications with SharpShooter Collection components it is recommended to use high-level integrated development environment as Microsoft Visual Studio 2005, Microsoft Visual Studio 2008, Microsoft Visual Studio 2010, Borland Delphi 8, Borland C# Builder, etc.



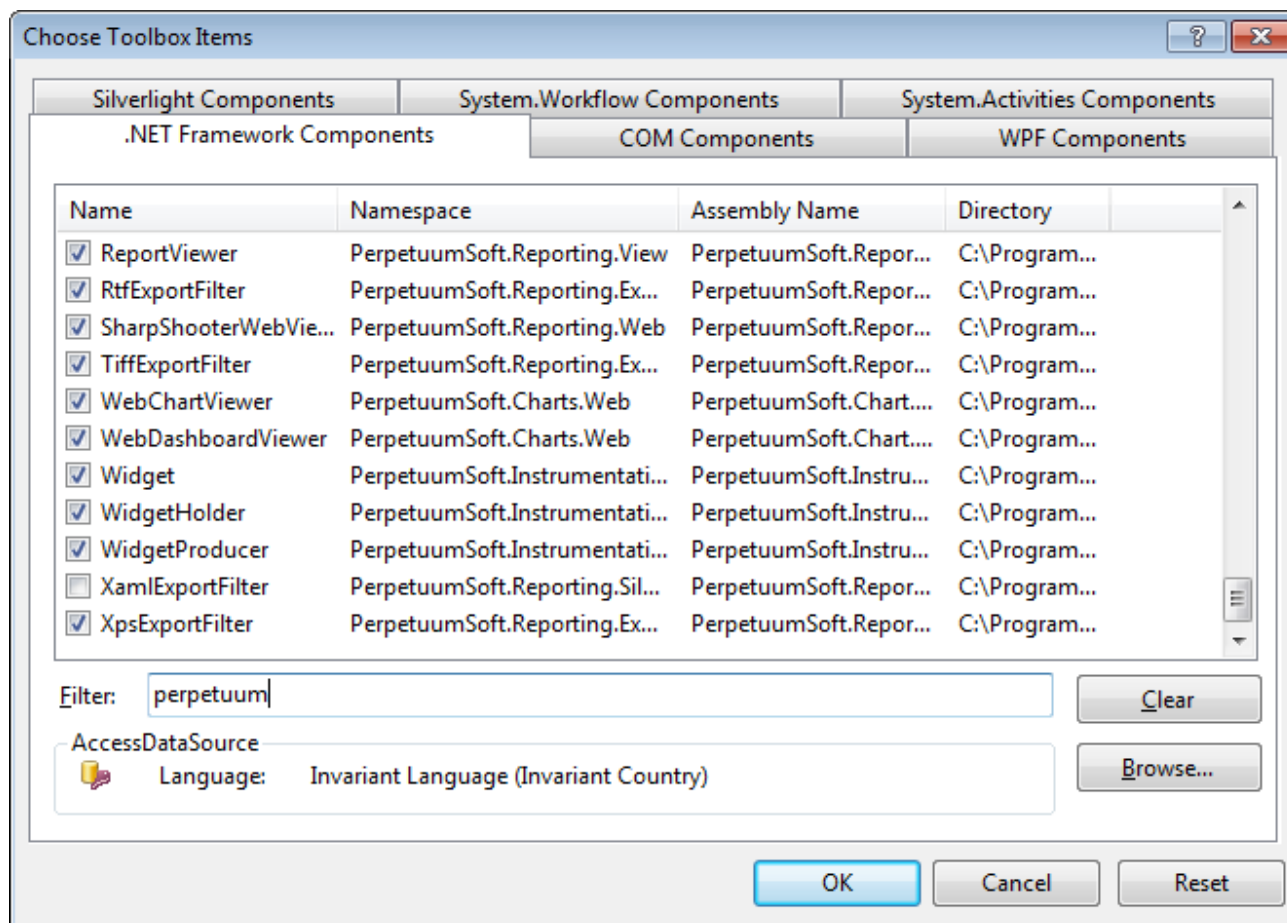
Microsoft .Net Framework 2.0 or higher is required for a successful installation. Microsoft .Net Framework 2.0 or higher is necessary for SharpShooter Reports.Silverlight usage. Minimal system requirements match Microsoft .Net Framework 2.0 requirements.

## Installation

SharpShooter Collection is supplied as a Microsoft Installer package. Package file name is SharpShooterCollection.msi. Run installation file and follow wizard instructions.

During the installation process one can add necessary components included into the product, add components to Visual Studio Toolbox ("Add components to Visual Studio Toolbox" button). This option is available in Microsoft Visual Studio 2005, Microsoft Visual Studio 2008 and Microsoft Visual Studio 2010. Otherwise (e.g. when using Microsoft Visual Studio Express Edition), one should add components manually. Let us examine the procedure of adding components in Microsoft Visual Studio.

- 1) Add a new tab to a toolbox using "Add tab" item of a contextual menu.
- 2) Select the added tab and click "Choose Items" menu of a toolbox.
- 3) Apply filter "Perpetuum" in order to find elements included into the product. All the components included into the SharpShooter Collection have "Perpetuum" word in the beginning of the name.
- 4) Check the components that are necessary for the complete functionality of SharpShooter Reports. If you need to use the product with WinForms you'll need the following components: ReportManager, DesignerDataSourcesTree, DesignerDocumentTree, DesignerErrorList, DesignerPropertyGrid, DesignerStatusBar, DesignerToolBar, DesignerToolBox, ReportDesigner, CsvExportFilter, ExcelExportFilter, ExcelXmlExportFilter, BitmapExportFilter, EmfExportFilter, GifExportFilter, JpgExportFilter, PngExportFilter, TiffExportFilter, HtmlExportFilter, PdfExportFilter, RtfExportFilter, XpsExportFilter, ReportViewer. SharpShooterWebViewer is used to display reports as Web pages. ReportViewer component included into PerpetuumSoft.Reporting.Silverlight.Client is used for making and displaying reports using Silverlight.



5) Press "Ok" and checked elements will be added to the current tab.

If you use Windows Vista or Windows 7 with UAC enabled, run setup file as administrator. In order to do this, run CMD as administrator and then run setup file from it.



## Editions

SharpShooter Reports includes the following products:

SharpShooter Reports.Win – functionality allows building and displaying reports in Windows Forms. Contains ReportViewer and ReportDesigner - end-user report designer that allows editing report templates and final documents.

SharpShooter Reports.Web – functionality allows building and displaying reports in Web applications. The product is the true cross platform report viewer as it offers the ability to deliver reports to Windows, Linux, Mac OS or even iOS and Android tablets and phones.

SharpShooter Reports.Silverlight – offers developer functionality for building and displaying reports in Silverlight applications. The edition contains ReportViewer - the component for report viewing in Silverlight applications.

SharpShooter Reports.Enterprise – allows building and displaying reports in Windows Forms, Web and Silverlight applications. The package includes all the necessary components: SharpShooter Reports.Win, SharpShooter Reports.Web and SharpShooter Reports.Silverlight.

SharpShooter Reports.Professional – edition allows building and displaying reports as Windows Forms, Web and Silverlight apps and contains complete source code.

## Deploy

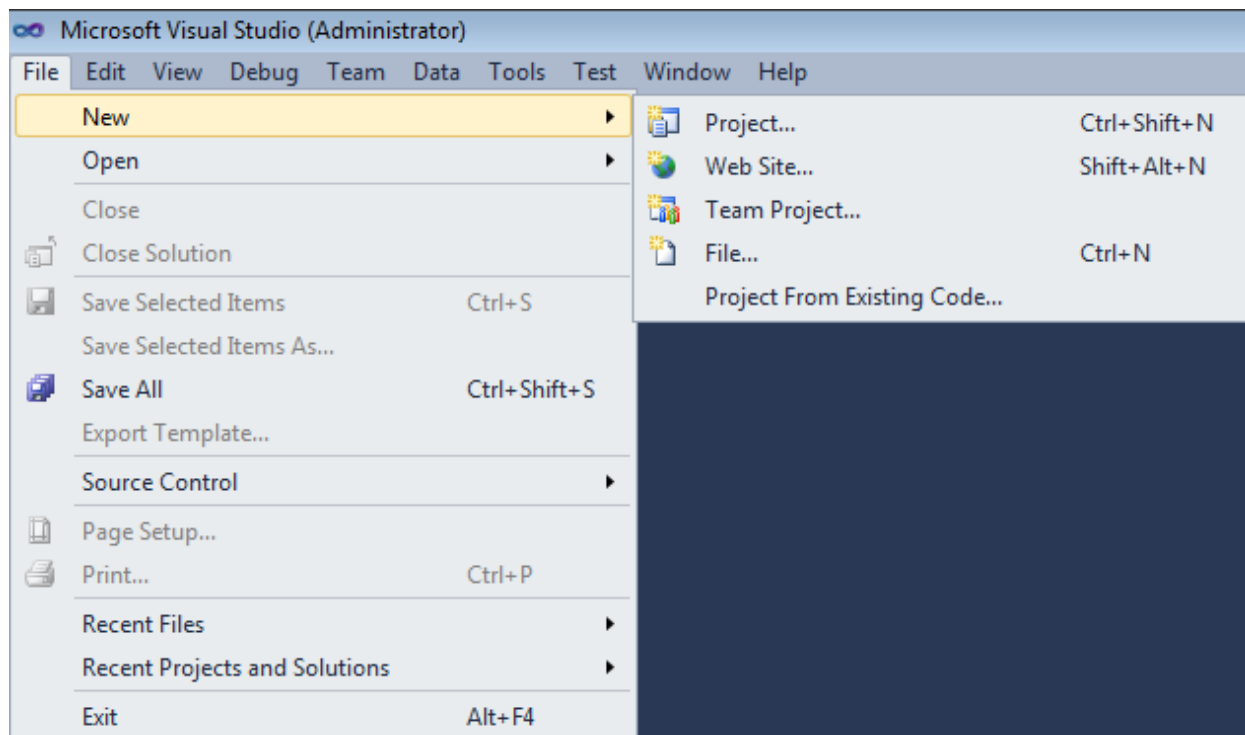
You don't have to do anything special to deploy a project using our product components. All you got to do is to copy compiled files into a folder in a computer file system or a web site. There is no need to install a product or license on a client computer. It is enough to have a licensed product on the developer's computer. License integrates into the product during compilation.



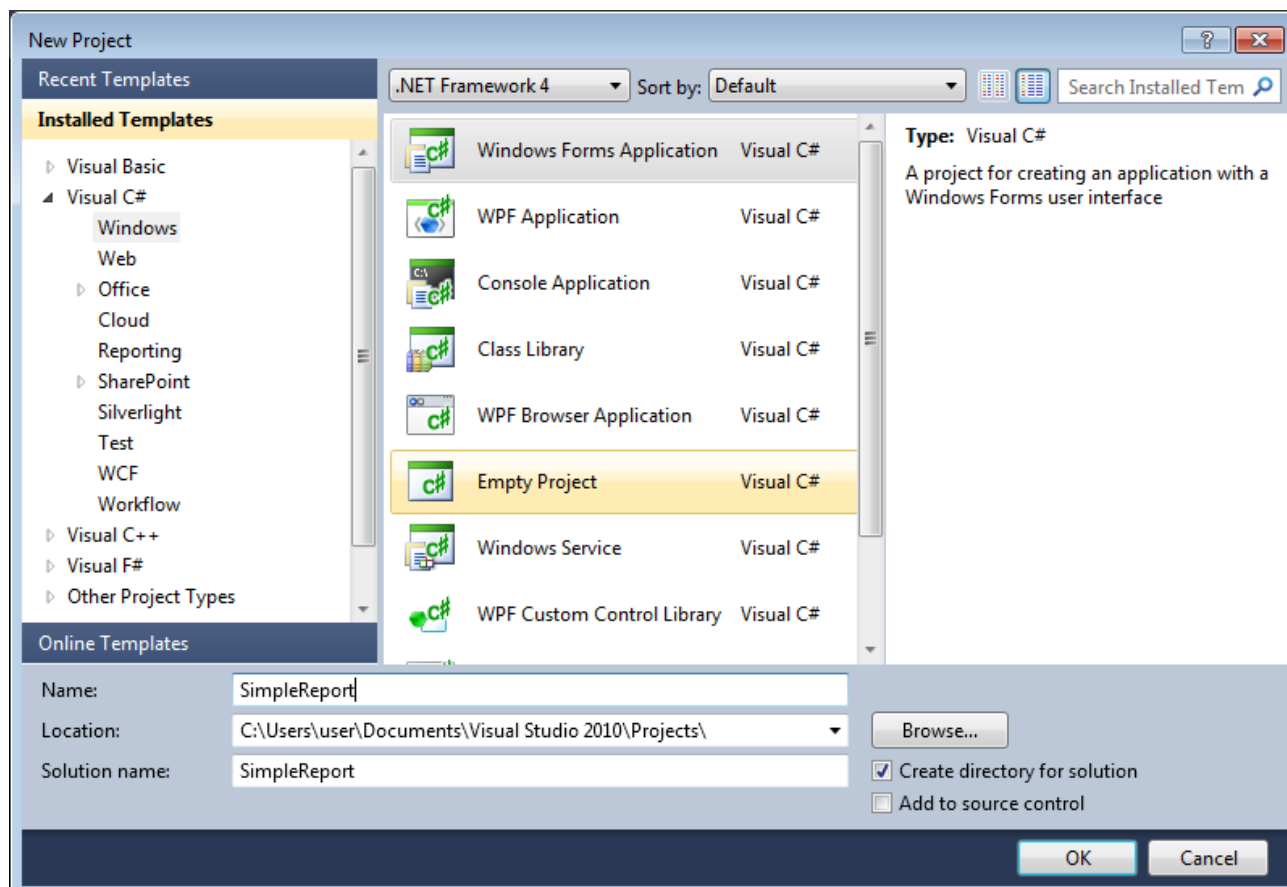
## Getting Started

Template of a report displaying current date, a logo and a note.

1. Create a new project in Microsoft Visual Studio. Select "New\Project" main menu item.



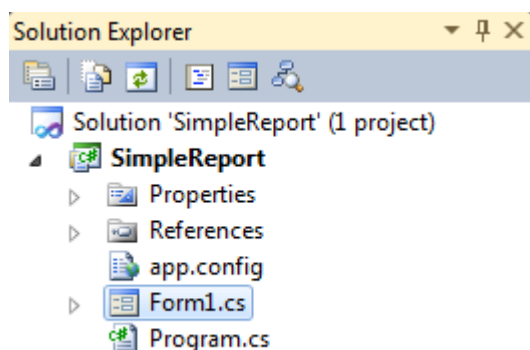
Select Windows Forms Application project template and set the name of a project – "SimpleReport", specify a folder for the project to be saved.



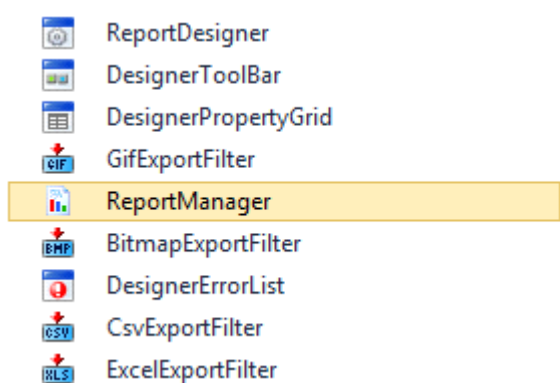




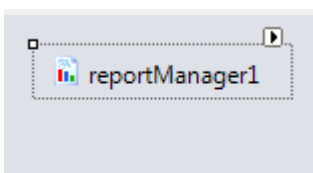
2. Open the main form of the application in the editor window with a double click on "Form1.cs" in Solution Explorer.



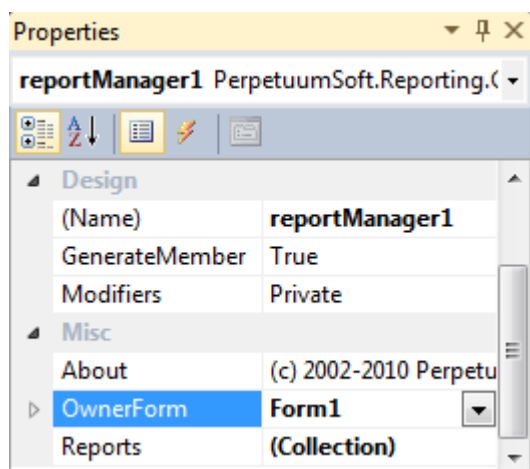
Drag and drop "ReportManager" element from the Toolbox. This element stores collections of report templates and data sources.



The component will be displayed in the lower part of a window.

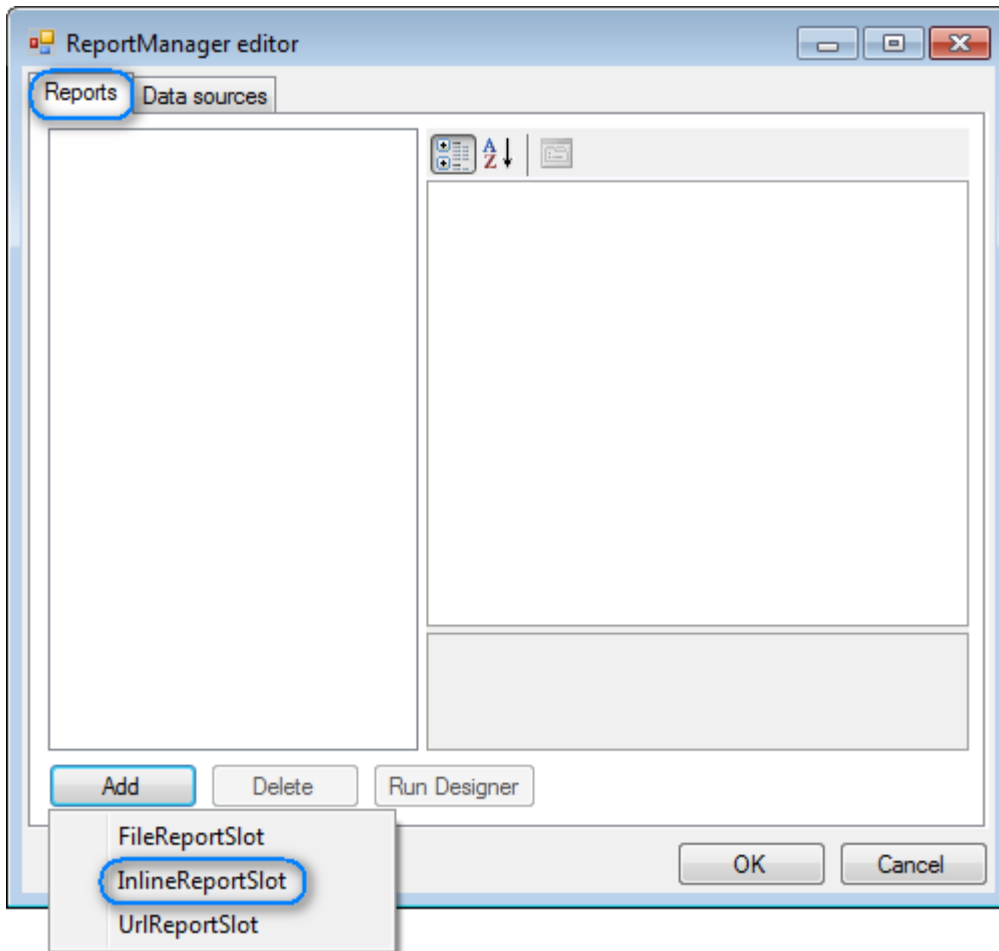


3. On the property grid, initialize the property OwnerForm of the ReportManager component by selecting the form it is located on.





4. Double-click "ReportManager" component to open ReportManager editor.

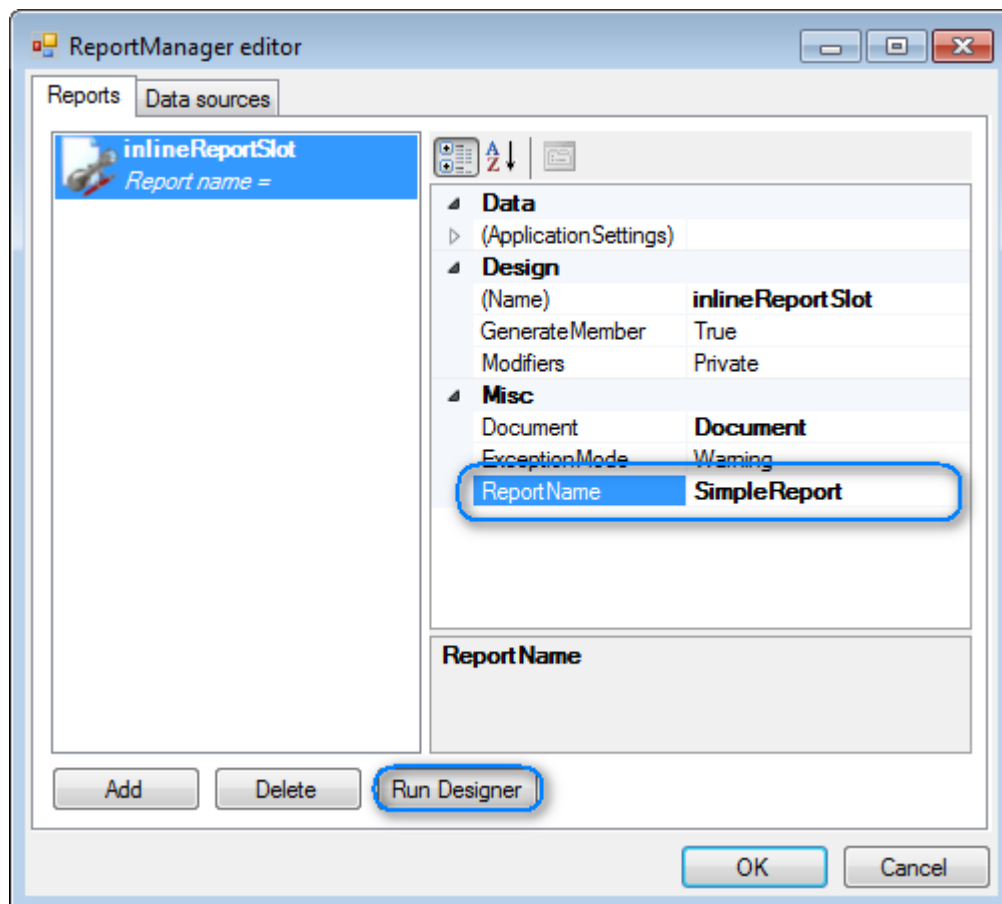


Press "Add" button on the "Reports" tab and select "InlineReportSlot".

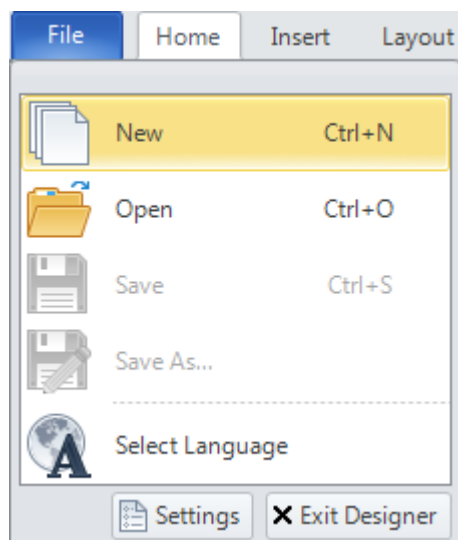


5. Set report name in a ReportName property – “Simple Report”.

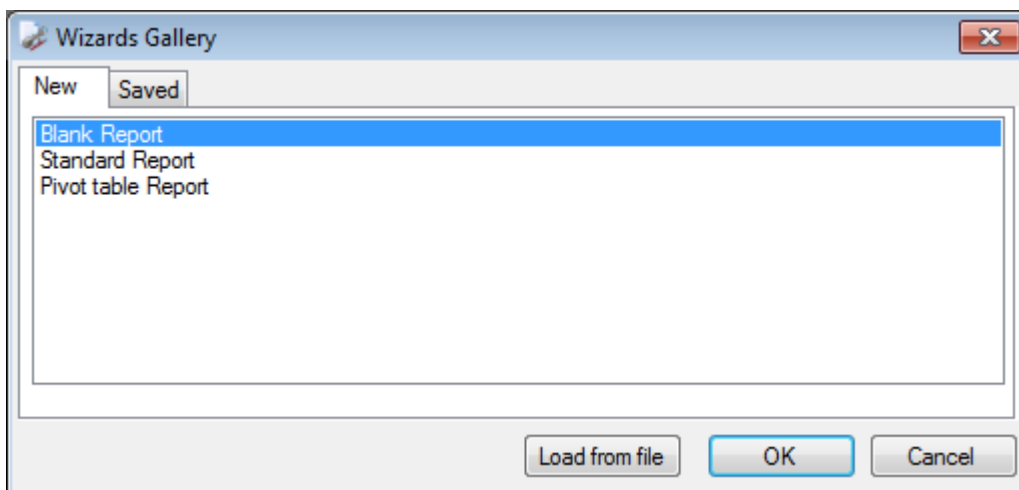
Press “Run Designer” button to open Report Designer template editor.



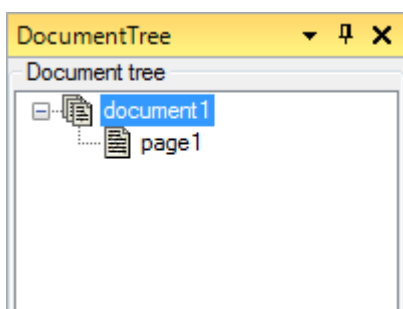
6. Create a new empty template – select “New” item in the Application menu.



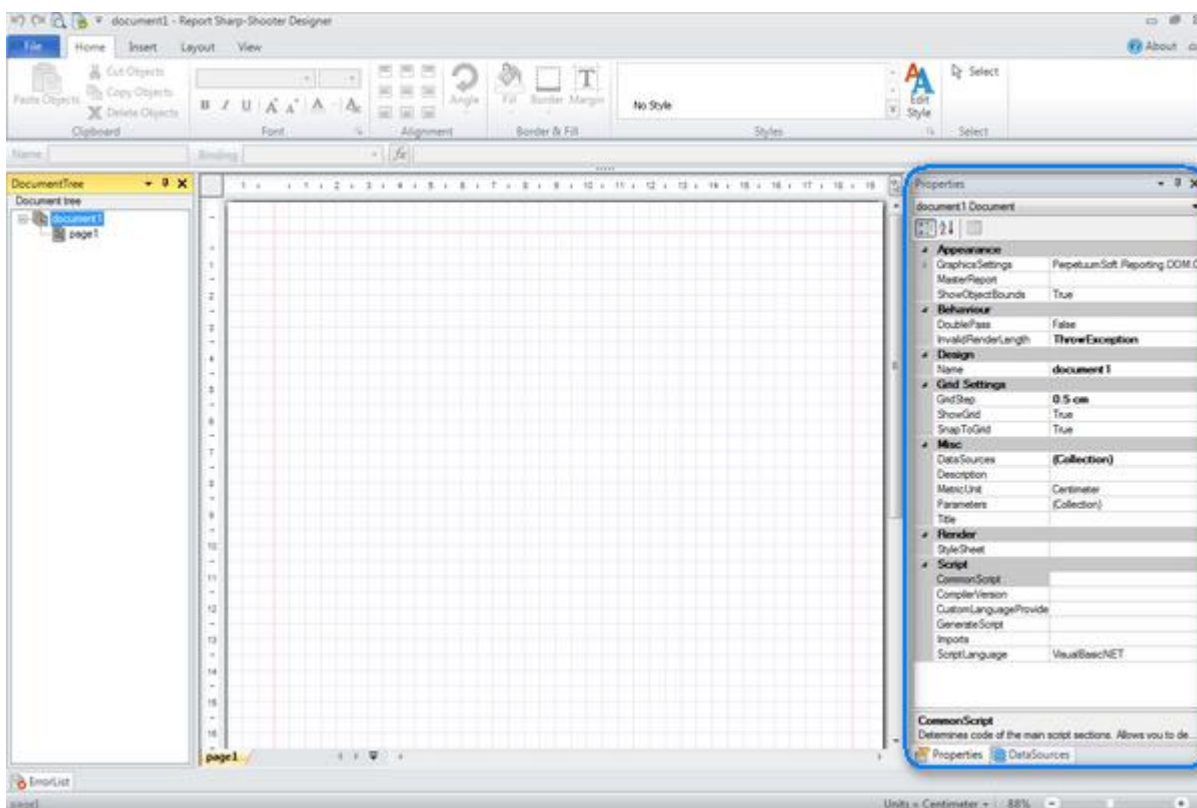
Select “Blank Report” item in the Wizards gallery and press “OK”



7. Select "document" in the Document Tree.

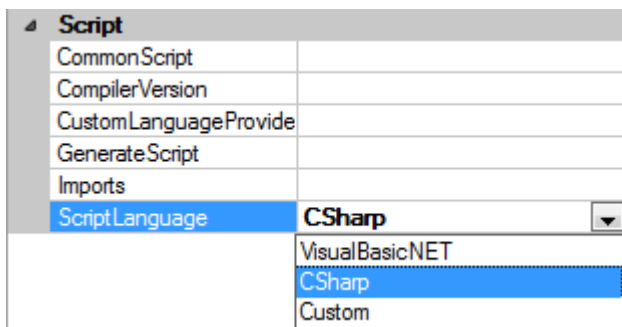


You will see properties of the edited template on the "Properties" window.

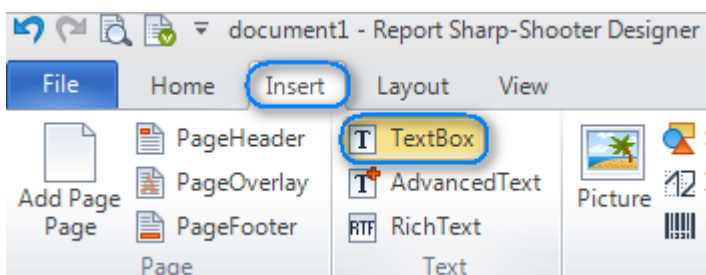




Set ScriptLanguage property to "CSharp".

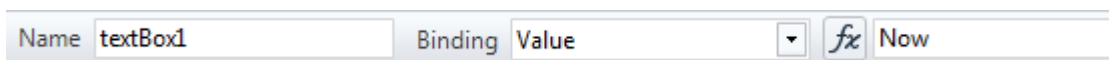


8. Go to the "Insert" tab. Click "TextBox" button.

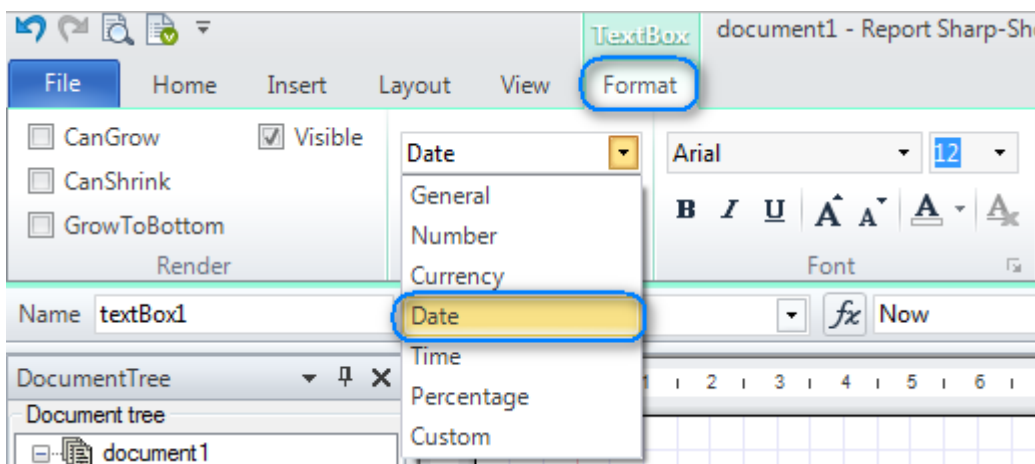


Left-click to add TextBox to the template.

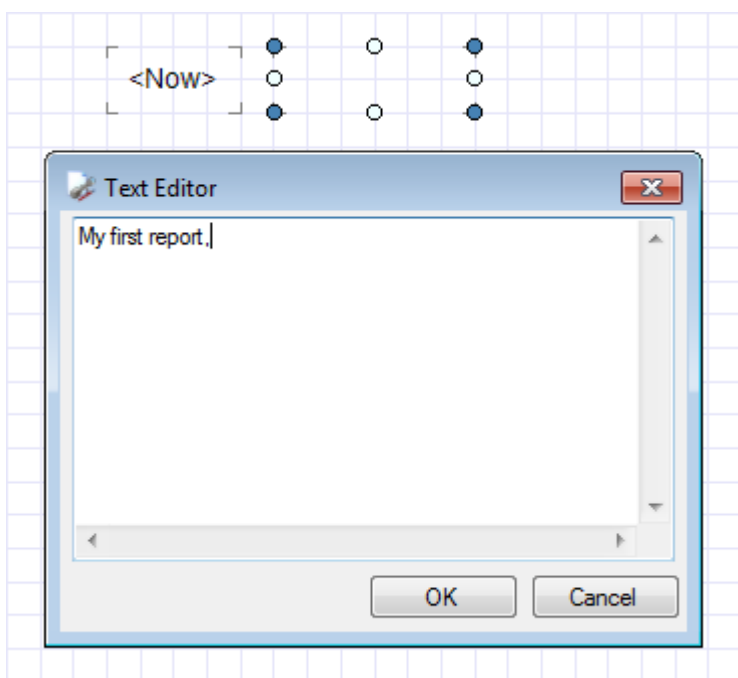
Set Value property to "Now".



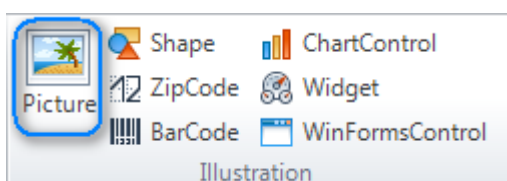
Go to "Format" tab of TextBox tab set. Select "Date" value in the TextFormat list.



9. Add one more TextBox element to the template. Open Text Editor (by double click on the TextBox or by pressing Enter key). Type "My first report".



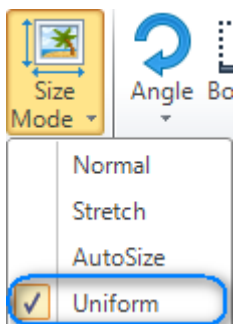
10. Go to the "Insert" tab. Click "Picture" button.



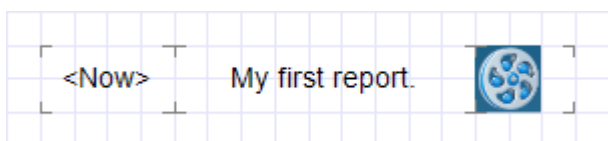
Left-click on the template to add Picture element

Double-click Picture element area to open dialog window setting path to the image. Choose a logo image and press "Open" button.

Go to "Format" tab of Picture tab set. Click "Size Mode" button and choose "Uniform" property value.

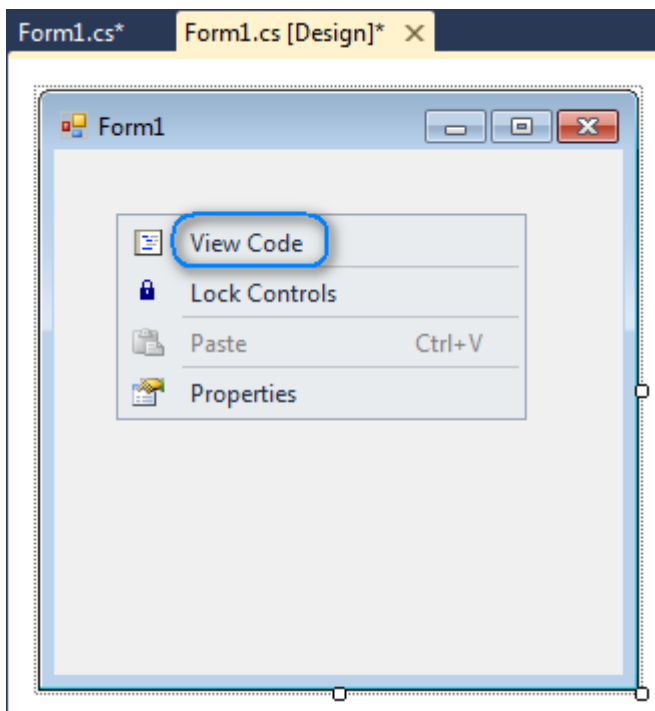


Report Template:



11. Save the template and close Report Designer.

12. Right-click on the form and select “View code” item of a context menu in order to switch to the code.



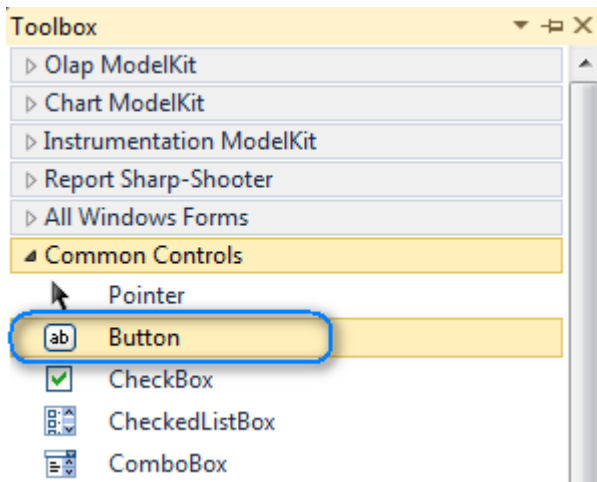
Add report displaying code to the class constructor. Create RenderCompleted event handler of the InlineReportSlot object.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        inlineReportSlot.RenderCompleted += new EventHandler(reportSlot_RenderCompleted);
        inlineReportSlot.Prepare();
    }
    private void reportSlot_RenderCompleted(object sender, EventArgs e)
    {
        using (PerpetuumSoft.Reporting.View.PreviewForm previewForm = new
PerpetuumSoft.Reporting.View.PreviewForm(inlineReportSlot))
        {
            previewForm.WindowState = FormWindowState.Maximized;
            previewForm.ShowDialog(this);
        }
    }
}
```

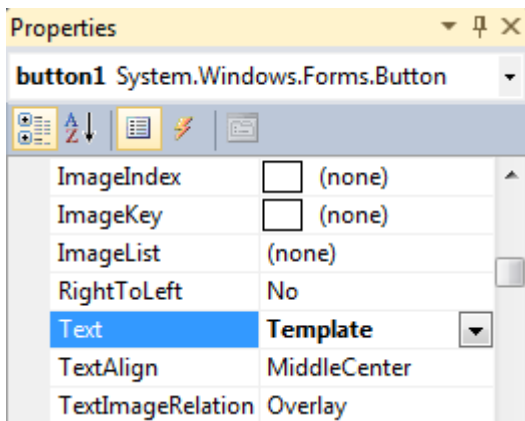


12. Switch back to the application form – click Form1.cs[Design] tab.

Add two buttons onto the form (drag and drop “Button” element from the Toolbox to the form).



Select “Button” element on the form, change Text property. Set Text = Template for the first button and Text = Report for the second one.



Create Click event handlers for the buttons – double-click Button element on the form. Add code that starts report generation to the handler. E.g., the following code:

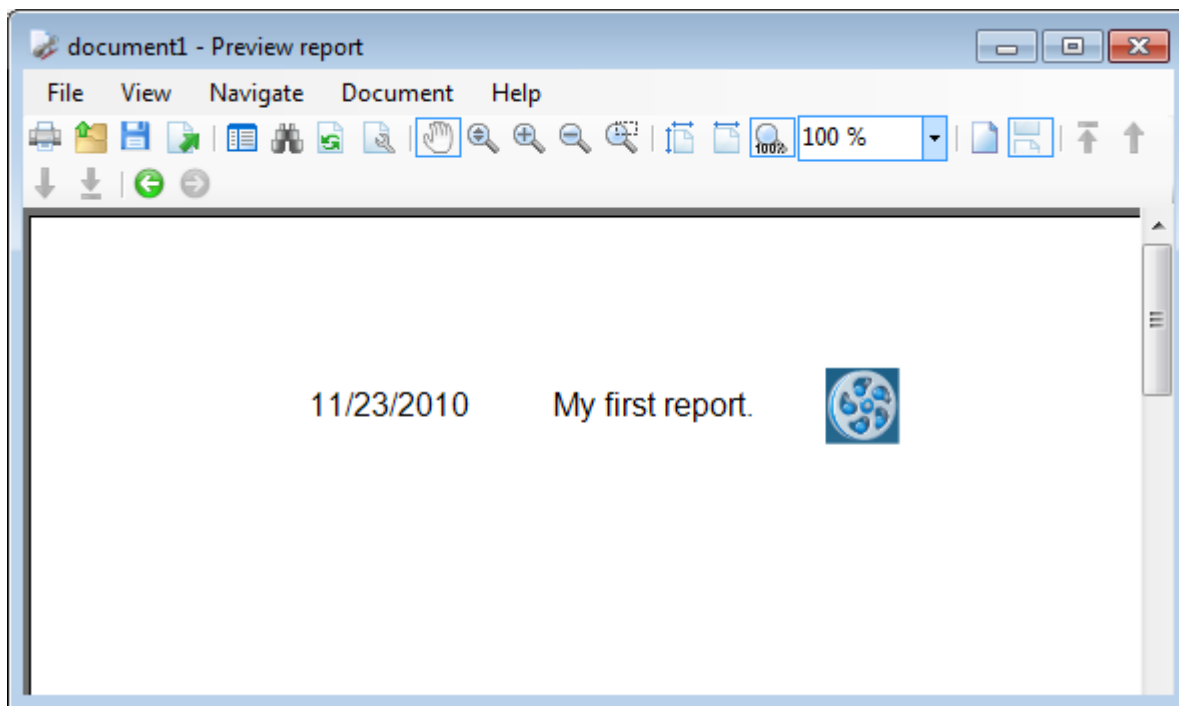
```
private void button1_Click(object sender, EventArgs e)
{
    inlineReportSlot.DesignTemplate();
}

private void button2_Click(object sender, EventArgs e)
{
    inlineReportSlot.Prepare();
}
```

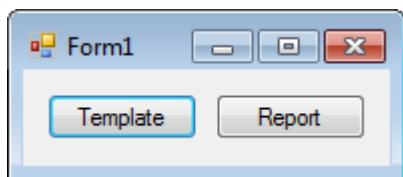
13. Press “Start Debugging” button on the Visual Studio Toolbar to run the application.

Generated report in Report Viewer.





The "Template" button opens template with Report Editor for editing. The "Report" button generates report and opens it with Report Designer.





## Basic Information

### Basic Terms

Let's consider basic terms used to work with SharpShooter Reports.

#### *Report Template*

Report Template is an object of the PerpetuumSoft.Reporting.DOM.Document class with the IsTemplate property set to "True", containing one or multiple pages containing some elements. Appearance of the final report is set by layout of these elements in a respective way and initialization of their properties. For example, you set report data sources to get data for reports and the way this data is displayed. Template contains scripts that are executed during report generation. Template file are of xml format and .rst extension; you can get it by saving template from the Report Designer. Objects of the PerpetuumSoft.Reporting.Components.ReportSlotclass are used to store report template. LoadReport and SaveReport methods of the ReportSlot objects are used to get and save report template.

(More information about report template)

#### *Report Designer*

Report Designer is a component used to create and edit templates. It provides the ability to set template properties and template page properties, add objects, set their layout and properties, set report data sources and to define report appearance and content of the report. With Report Designer it is possible to save template or open saved templates for editing and forming reports. It is possible to call Report Viewer from the Report Designer to view formed reports.

(More information about Report Designer)

#### *ReportManager*

ReportManager is a component to store report and data source collections. It is possible to access objects collection storing report templates via the Reports property; data sources collection is available via the DataSource property.

#### *Final Document*

Final Document is an object of the PerpetuumSoft.Reporting.DOM.Document class with the IsTemplate property set to False. It contains only those elements that contain visible information such as text, charts, gauge or pictures. Final document file is of xml format and .rsd extension.

#### *Report Viewer*

Report Viewer is a component used to view final document. The following manipulations over the document are available: zoom, search, bookmark tree, various view parameters. It is possible to export, print and save reports to a file. To edit final document it is necessary to run Report Designer.

(More information about Report Viewer)

#### *Export Filter*

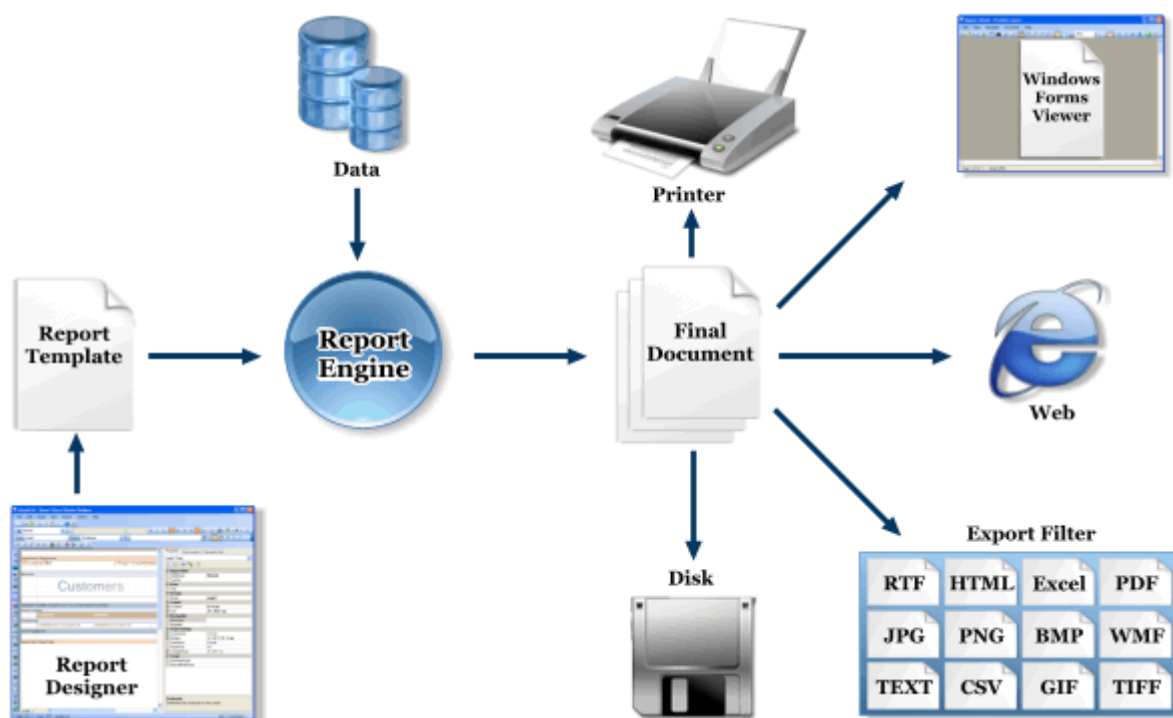
SharpShooter Reports provides the ability to export final document to many popular formats as GIF, PNG, JPG, BMP, EMF, PDF, HTML, CSV, TXT, Excel, XML, XML for Excel, RTF.

(More information about export filters)



### Interaction Schema

Below is a schema of the report life cycle.



### Main Components Used to Create a Report

Appearance of the final document is defined by the objects contained in the report template. Every object has a set of properties; when you set them you define report appearance. These properties can be set by a constant value or expression that will be calculated during report rendering. These are binding scripts.

All objects used to create a report can be divided into two classes: bands and visual elements.

Bands are not displayed in the final document; they are containers for other elements and are designed to set the way of displaying elements they contain: form lists, tables, set data sources, set grouping and sorting conditions. Some bands can be containers for other bands.

(More information about bands)

Visual elements of the report are used to display data of various types: text, pictures, zip and bar codes, shapes, image of the Windows Forms controls. Properties of the visual elements set format and style of data.

(More information about visual elements)

### Using Data Sources

ADO.NET objects, custom classes implementing System.ComponentModel.IListSource of System.Collection.IEnumerable interface and other data sources. In the last case, properties of the passed object will be used report data.

It is possible to set data sources using ReportManager; such data sources will be available for all report templates stored in this ReportManager. Besides, data sources can be built in the certain template.

Built-in data sources can be of three types: SqlDataSource, XmlDataSource and OdbcDataSource.

(More information about data sources)



## Using Scripts

Scripts are used to set expressions that are calculated during report generation. Result of expressions calculation can be used as property values. Scripts can be used to generate element in the final report and manual layout of report elements.

Scripts are expressions on some programming language. It is possible to use any .Net programming language.

It is possible to create and use instances of any classes available for the application using SharpShooter Reports during report generation.

(More information about scripts)

## Binding Elements to Data

Data binding is used in reports displaying data from the data source and it is executed in two passes. At first, object that will be used as a data source is set. Then field for the certain element is set. If report data source is a collection element, collection element field is set; and an element nested inside DataBand is used to display list. If the object is not a collection, report can display its properties. Binding final elements to data sources is executed by the binding expressions.

(More information about data binding)

## Document Generation

To start report generation process the following methods of the ReportSlot objects are used: Prepare and RenderDocument. The difference between these two methods is that Prepare method executes report generation process in a separate thread. ReportManager has OwnerForm property used to set thread of the form report generation events will be executed in. Initialization of this property may be required if event handlers refer to elements located on the form. Execution of such handlers in a separate thread may result in System.InvalidOperationException.

In order to control and have the ability to declare about report rendering process, it is possible to use overloads of the Prepare and RenderDocument methods; these overloads get object as a parameter and implement IRenderSite interface. Object implementing this interface can be used to get information on report generation process. It is just necessary to set object methods described in the interface.

If the template uses elements that get data from the data source, data source with the set in the element name is searched during report generation. Search is executed among sources added to ReportManager that is included in the current ReportSlot and among built-in data source. Besides, when you get data source, event is called; you can handle it to dynamically connect to a data source.

Report parameters may be required during report generation. GetReportParameter event is used to get parameters. It is necessary to initialize collection of named parameters in its handler: this collection can be got via Parameters property of the PerpetuumSoft.Reporting.Components.GetReportParameterEventArgs object. Instance of this class is delegated to the handler as a parameter. Please pay attention to the fact that if parameter values are got from the controls properties, it is necessary to initialize OwnerForm property of the corresponding ReportManager.

If SubReport is used in the report, the corresponding ReportManager should contain ReportSlot with the ReportName property matching SubReport.TemplateName property. If the nested template is created dynamically, it is possible to use ResolveSubReport event of the ReportManager to initialize it. PerpetuumSoft.Reporting.Components.ResolveSubReportEventArgs object is passed to this handler as a parameter. This object has a TemplateName property; this property defines what nested report is necessary and passes it via the Template property.

(More information about managing report generation)



### Visualization and Editing Final Documents

In order to view final document it is possible to use Windows Forms component – ReportViewer; to view reports from ASP.NET applications use SharpShooterWebViewer.

Final document consists of only visual elements generated and located in the report in accordance with report template. Final document can be opened in the Report Designer to edit all static properties of the elements: size, layout, style. It is also possible to add new visual elements to the document. When you add bands, they will be displayed in the final document in the way they are displayed in the Report Designer.

### Export, Print and Saving

After the creation final document can be delivered to the customer in various forms. SharpShooter Reports offers report export to various formats: PDF, GIF, HTML, XML for Excel, RTF, etc. There are some options to set export parameters, for example, picture quality settings, selecting certain page of the document, etc.

Final document can be saved to hard disk. Document is saved in the xml format with .rsd extension. .rsd files are associated with the Report Viewer.

It is possible to print document from the Report Viewer. Print settings specify numbers of the printed pages and the amount of document copies.



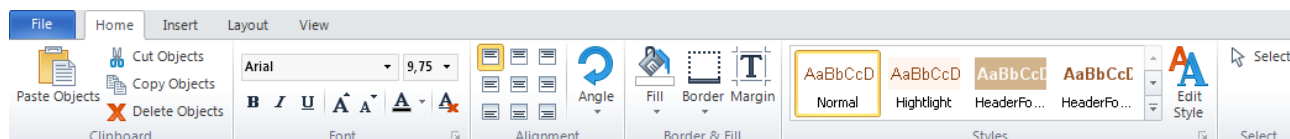
## Report Designer Interface

Report designer is designated to create and edit report templates. In the designer, you can add pages, and objects to the template, set their static and dynamic properties, and connect to data sources.

### Tabs

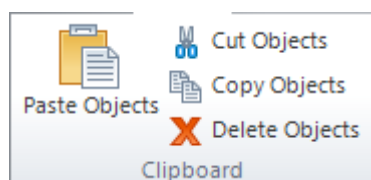
#### Home Tab

This tab contains the most frequently used commands.



#### Clipboard Group

This group contains buttons for working with the clipboard.



The "Paste Objects" button pastes the contents of the Clipboard into the selected area.

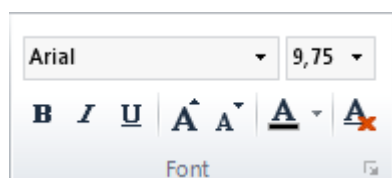
The "Cut Objects" button cuts the selected objects and put it on the Clipboard.

The "Copy Objects" button copies the selected objects and put it on the Clipboard.

The "Delete Objects" button deletes the selected objects.

#### Font Group

This group contains controls for working with Font property.



The "Font Name" list contains available styles (sets the Font.FamilyName property).

The "Font Size" list contains size values and allows the setting of arbitrary size (sets the Font.Size property).

The "Bold" button sets bold font for the text (sets the Font.Bold property).

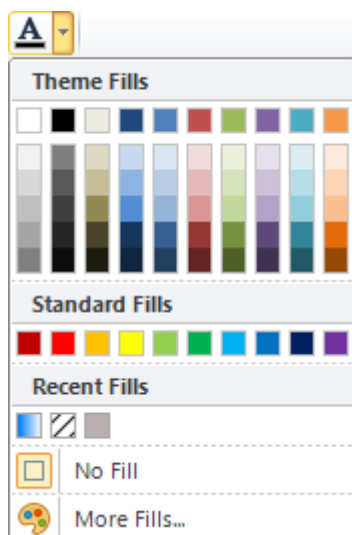
The "Italic" button sets italic font for the text (sets the Font.Italic property).

The "Underline" button sets underlined font for the text (sets the Font.Underline property).

The "Increase Font Size" button increase font size (sets the Font.Size property).

The "Decrease Font Size" button decrease font size (sets the Font.Size property).

The "Text Fill" button sets color and fill for text (sets the TextFill property of the selected object).



The "Theme Fills" and "Standard Fills" sections contain color palette.

The "Recent Fills" section contains fills, which were recently used.

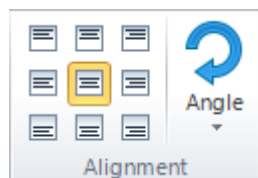
The "No Fill" menu item cancels all fill settings and sets TextFill property to "[None]".

The "More Fills..." menu item opens "Fill Editor" for creating custom fill.

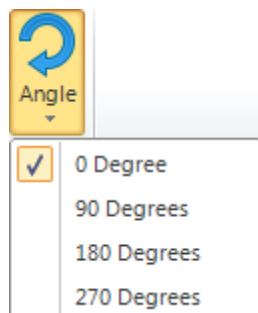
The "Use Style Only" button sets the style settings. These settings correspond to the selected style and remove all changes.

### Alignment group

The "Alignment" table allows setting TextAlign and ImageAlign properties.



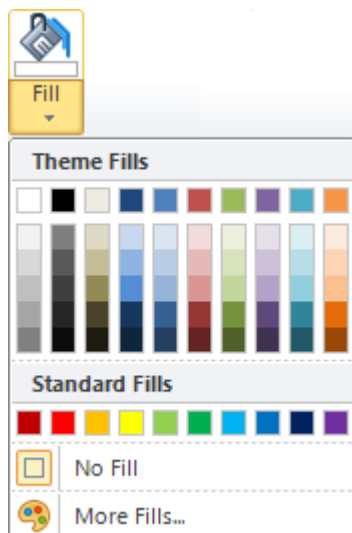
The "Angle" button sets Angle property, opens the list of allowed property values.



## Border & Fill Group



The "Fill" button sets color and fill for background (sets the Fill property).



The "Theme Fills" and "Standard Fills" sections contain color palette.

The "Recent Fills" section contains fills, which were recently used.

The "No Fill" menu item cancels all fill settings and sets Fill property to "[None]".

The "More Fills..." menu item opens "Fill Editor" for creating custom fill.

The "Border" button sets border (sets the Border property).



**1** – This menu items set borders lines location.

**2** – This submenu items set border lines style and color: the "Line Color" submenu contains color palette; the "Line Style" submenu contains line types.

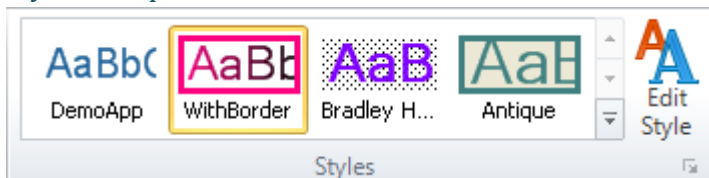
**3** – The "More Borders..." menu item opens "Border Editor".

The "Margins" button opens "Margins Editor" for setting Margins property.

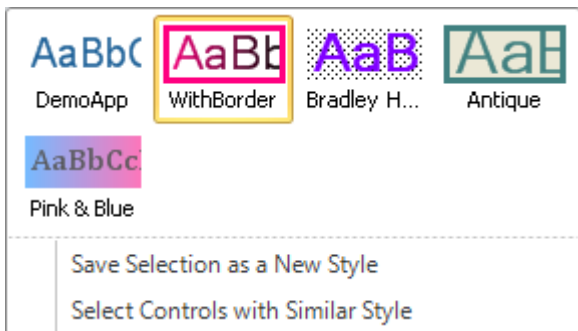




### Styles Group



The "Styles Gallery" includes preview of the template's styles. Click on style's preview applies this style to the selected object (sets StyleName property).



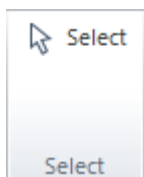
The "Save Selection as a New Style" menu Item opens "Create New Style from Formatting" dialog for setting name of new style and saves the style of selected object.

The "Select Controls with Similar Style" menu item selects all objects on the current page, which StyleName property is set to the style of selected object.

The "Edit Style" button opens "Style Sheet Editor".

### Select Group

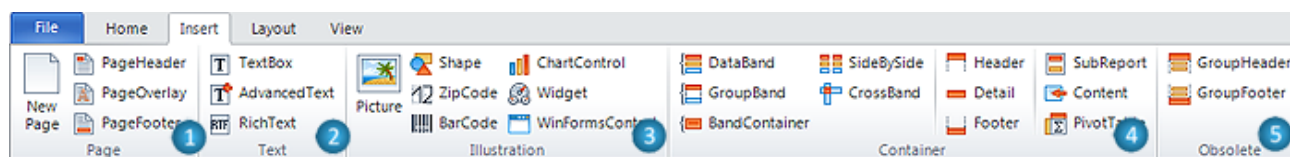
The "Select" button switches the Select Mode on.





## Insert Tab

This tab allows adding objects to a template. Buttons are divided into groups.



**1** – The “Page” group contains buttons, which allow adding new pages, PageHeader, Page Overlay and PageFooter bands.

**2** – The “Text” group contains buttons, which allow adding text output elements: TextBox, AdvancedText and RichText.

**3** – The “Illustration” group contains buttons, which allow adding various visual elements: Picture, Shape, ZipCode, BarCode, ChartControl, Widget and WinFormsControl.

**4** – The “Container” group contains buttons, which allow adding various bands: DataBand, GroupBand, BandContainer, SideBySide, CrossBand, Header, Detail, Footer, SubReport, Content and PivotTable bands.

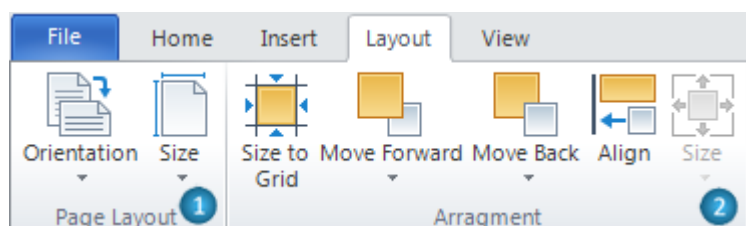
**5** – The “Obsolete” group contains buttons, which allow adding GroupHeader and GroupFooter bands, which are obsolete and needed for backward versions compatibility. But it is not recommended to use these components to create new reports. GroupBand together with Header and Footers should be used.

(Read more about Template Objects)



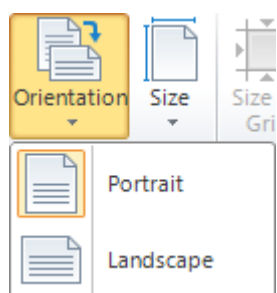
## Layout Tab

This tab contains controls that affect the page layout, spacing and positioning objects.

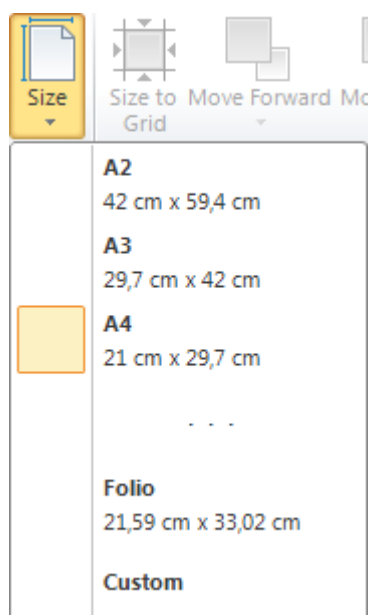


**1** – The “Page Layout” group contains buttons, which allow setting page properties:

The “Orientation” button sets Page.Orientation property. It opens the list of property’s values.



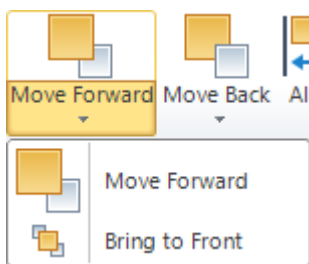
The “Size” button sets Page.PaperKind property. It opens the list of property’s values. The “Custom” menu item opens “Paper Size Editor” to set custom page size.



**2** – The “Arrangement” group allows management of object size and position:

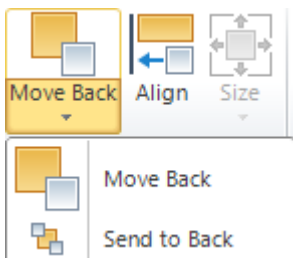
The “Size to Grid” button changes size of selected object to snap it to grid.

The “Align” button opens the “Align” Editor for objects alignment setting.



The "Move Forward" menu item brings the selected object forward so that it is hidden by fewer objects that are in front of it in the current container.

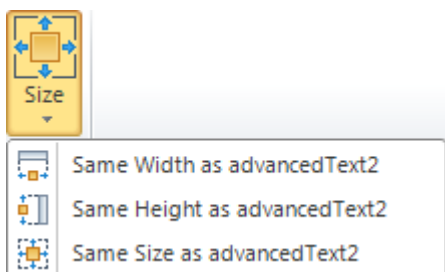
The "Bring to Front" menu item brings the selected objects in front of all other objects so that no part of it is hidden behind another object in the current container.



The "Move Back" menu item sends the selected objects backward so that it is hidden by the objects that are in front of it in the current container.

The "Send to Back" menu item sends the selected objects behind all other objects in the current container.

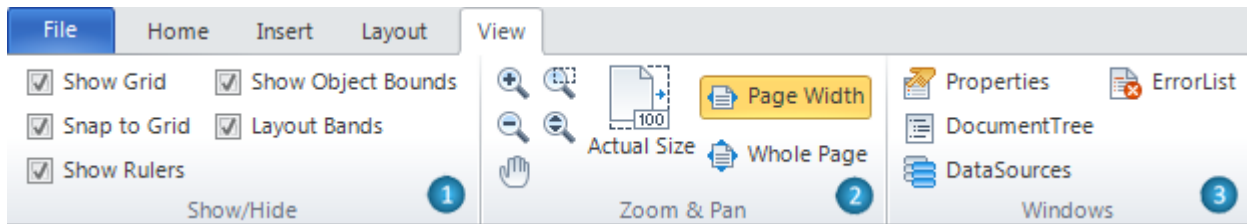
The "Size" button changes sizes of the selected objects. Menu items set the parameter as main selected object.





## View Tab

This tab contains controls that affect the template view, including view mode, show/hide options, zooming and window management.



**1** – The “Show/Hide” group allows setting designer view:

The “Show Grid” checkbox switches on/off the display of a grid (sets the Document.ShowGrid property).

The “Snap to Grid” checkbox switches on/off the option of binding to a grid (sets the Document.SnapToGrid property).

The “Show Rulers” checkbox switches on/off the display of a rule.

The “Show Object Bounds” checkbox switches on/off the display of objects bounds (sets the Document.ShowObjectBounds property).

The “Layout Bands” checkbox switches on/off the option of automatic layout of the bands.

**2** – The “Zoom & Pan” group allows setting zoom and Pan Mode.

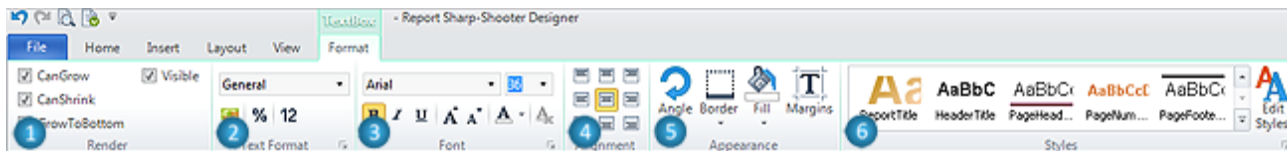
**3** – The “Windows” group contains buttons allows open Properties, Document Tree, Data Sources and Error List windows.

## Contextual Tabs

Contextual tabs are used to display a collection of commands that are relevant only when particular object is selected.

### TextBox Tab Set

The TextBox tab set includes "Format" tab.



The "Format" tab for TextBox includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, GrowToBottom and Visible.
- 2 – The "TextFormat" group sets TextFormat property. The list contains available property's values. The buttons allow set Currency, Percentage and Number format quickly.
- 3 – The "Font" group allows setting Font property.
- 4 – The "Alignment" group allows setting TextAlign property.
- 5 – The "Appearance" group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.
- 6 – The "Styles" group allows setting StyleName property.

### AdvancedText Tab Set

The AdvancedText tab set includes "Format" tab.

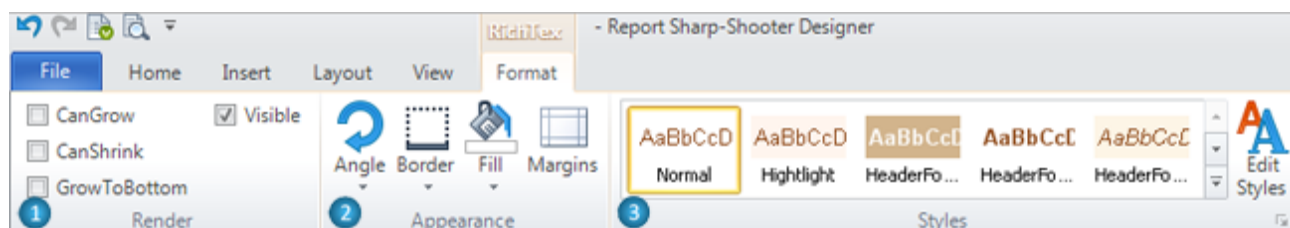


The "Format" tab for AdvancedText includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanBreak, CanGrow, CanShrink, GrowToBottom and Visible.
- 2 – The "TextFormat" group sets TextFormat property. The list contains available property's values. The buttons allow set Currency, Percentage and Number format quickly.
- 3 – The "Font" group allows setting Font property.
- 4 – The "Alignment" group allows setting TextAlign property.
- 5 – The "Appearance" group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.
- 6 – The "Styles" group allows setting StyleName property.

### RichText Tab Set

The RichText tab set includes "Format" tab.

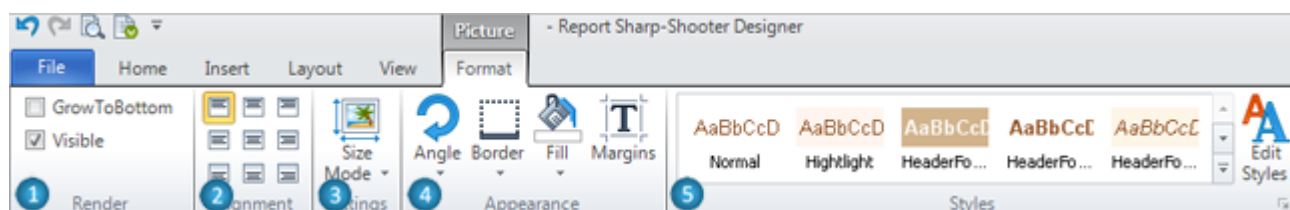


The "Format" tab for RichText includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, GrowToBottom and Visible.
- 2 – The "Appearance" group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.
- 3 – The "Styles" group allows setting StyleName property.

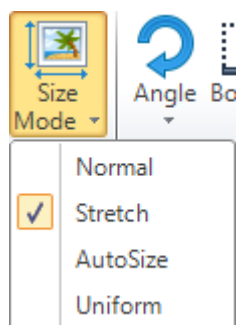
### Picture Tab Set

The Picture tab set includes "Format" tab.



The "Format" tab for Picture includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: GrowToBottom and Visible.
- 2 – The "Alignment" group sets ImageAlign property.
- 3 – The "Settings" group contains the "Size Mode" button, which sets SizeMode property, opens the list of allowed property values.

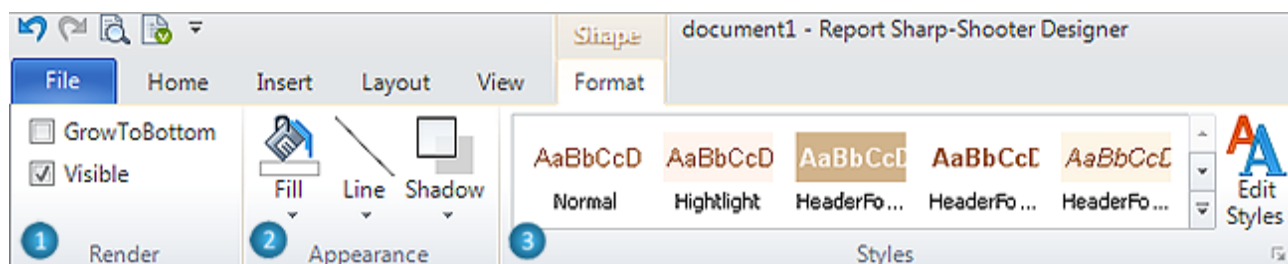


- 4 – The "Appearance" group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.
- 5 – The "Styles" group allows setting StyleName property.



### Shape Tab Set

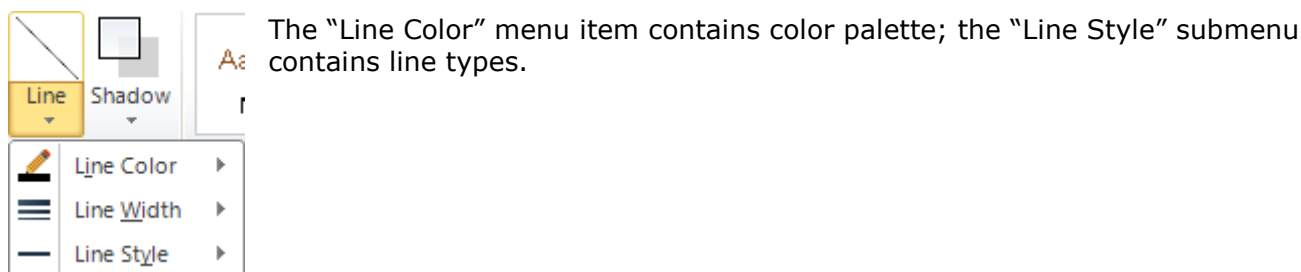
The Shape tab set includes "Format" tab.



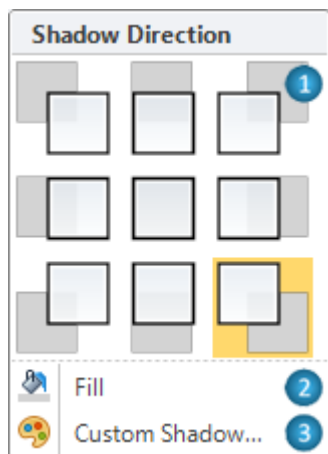
The "Format" tab for Shape includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: GrowToBottom and Visible.
- 2 – The "Appearance" group contains buttons for setting Appearance properties: Fill, Line and Shadow.

The "Line" button allows setting the line of shape: Line Color, Width and Style.



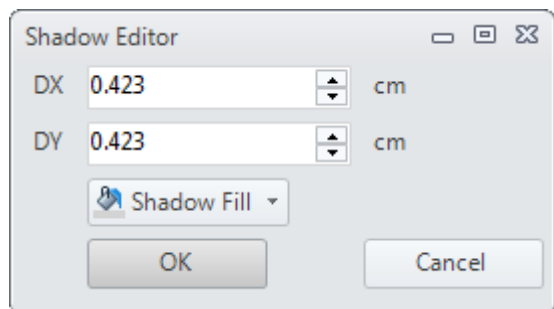
The "Shadow" button sets the shadow of shape.



2.1 – The area for setting shadow location.

2.2 – The "Fill" menu item opens color palette for setting shadow fill.

2.3 – The "Custom Shadow" menu item opens Shadow Editor for setting shadow properties.

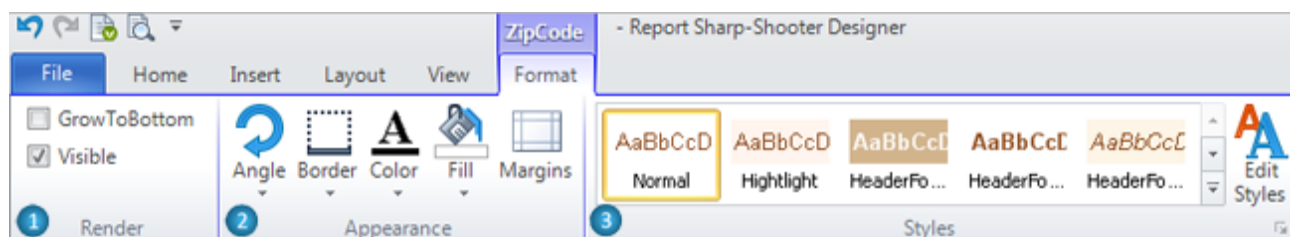


3 – The "Styles" group allows setting StyleName property.



### ZipCode Tab Set

The ZipCode tab set includes "Format" tab.

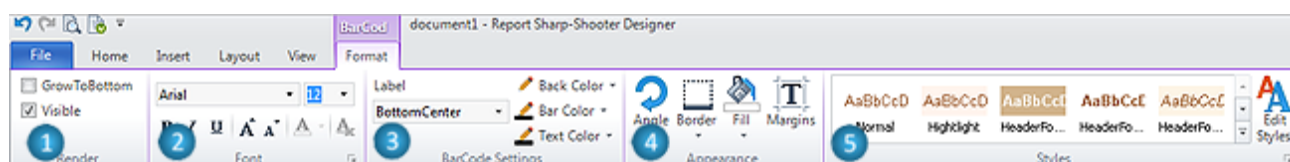


The "Format" tab for ZipCode includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: GrowToBottom and Visible.
- 2 – The "Appearance" group contains buttons for setting Appearance properties: Angle, Border, Fill, Margins and Color, which sets ZipCode.Color property.
- 3 – The "Styles" group allows setting StyleName property.

### BarCode Tab Set

The BarCode tab set includes "Format" tab.

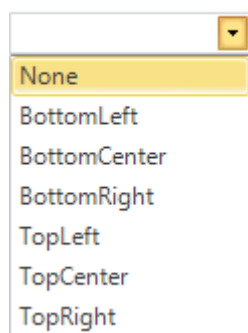


The "Format" tab for BarCode includes following groups:

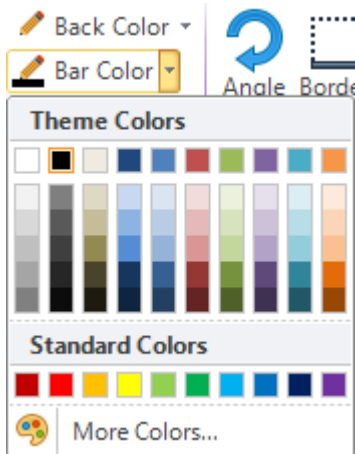
- 1 – The "Render" group contains Checkboxes for setting Render properties: GrowToBottom and Visible.
- 2 – The "Font" group allows setting Font property.
- 3 – The "BarCode Settings" group contains is for setting special BarCode's properties:

The "Label" list contains LabelPosition property values and "None" value for setting ShowLabel property to "False".

Label



The "Back Color", "Bar Color" and "Text Color" buttons set BackColor, BarColor and TextColor properties.

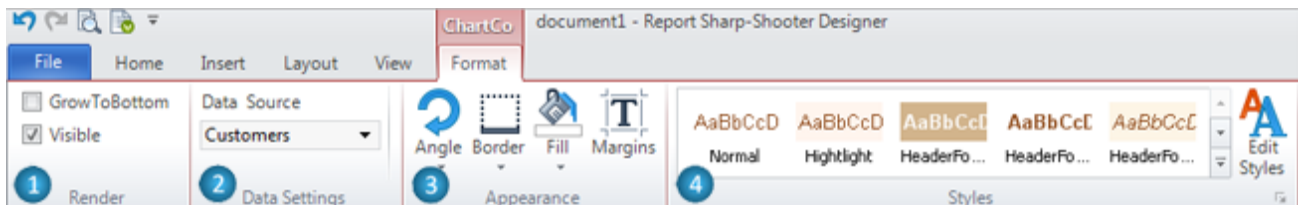


4 – The “Appearance” group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.

5 – The “Styles” group allows setting StyleName property.

### ChartControl Tab Set

The ChartControl tab set includes “Format” tab.



The “Format” tab for ChartControl includes following groups:

1 – The “Render” group contains Checkboxes for setting Render properties: GrowToBottom and Visible.

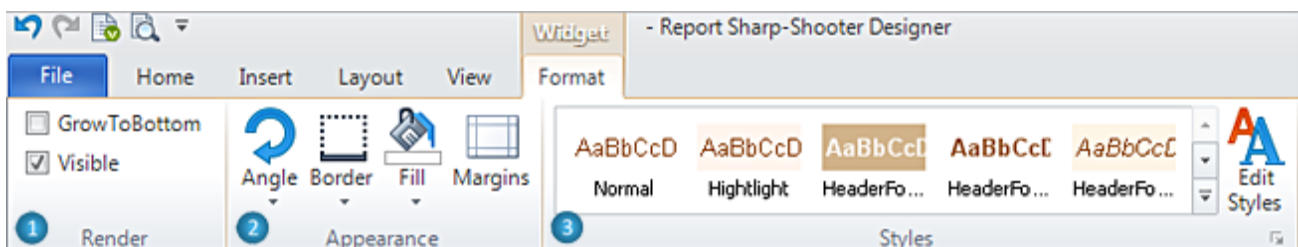
2 – The “Data Settings” group contains control for setting DataSource property.

3 – The “Appearance” group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.

5 – The “Styles” group allows setting StyleName property.

### Widget Tab Set

The Widget tab set includes “Format” tab.



The “Format” tab for Widget includes following groups:

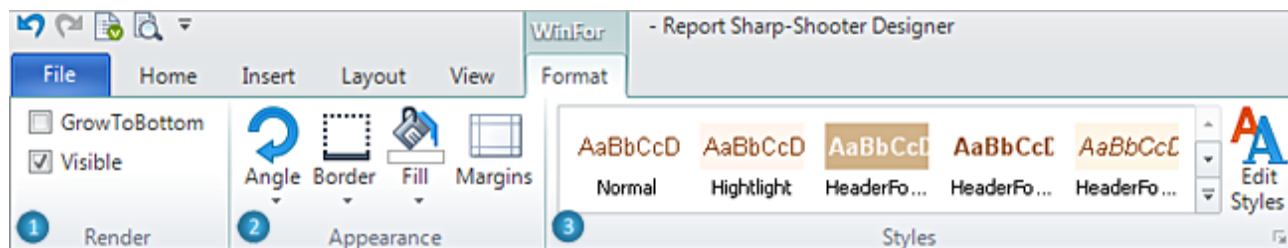
1 – The “Render” group contains Checkboxes for setting Render properties: GrowToBottom and Visible.

2 – The “Appearance” group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.

3 – The “Styles” group allows setting StyleName property.

### WinFormsControl Tab Set

The WinFormsControl tab set includes “Format” tab.



The “Format” tab for WinFormsControl includes following groups:

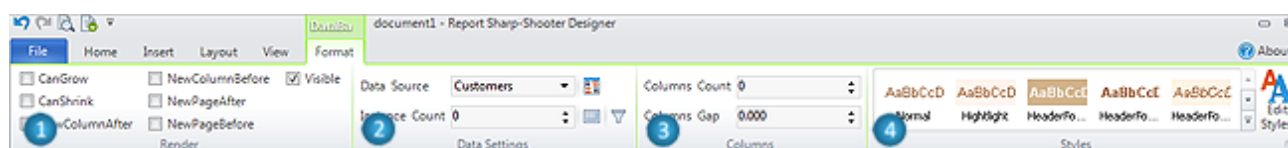
1 – The “Render” group contains Checkboxes for setting Render properties: GrowToBottom and Visible.

2 – The “Appearance” group contains buttons for setting Appearance properties: Angle, Border, Fill and Margins.

3 – The “Styles” group allows setting StyleName property.

### DataBand Tab Set

The DataBand tab set includes “Format” tab.



The “Format” tab for DataBand includes following groups:

1 – The “Render” group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.

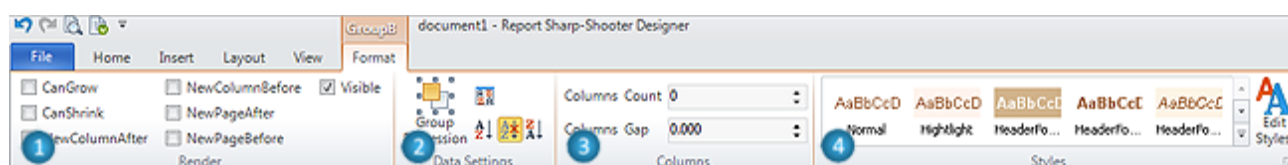
2 – The “Data Settings” group contains controls for setting following properties: DataSource, InstanceCount, Sort, Totals and FilterExpression.

3 – The “Columns” group contains controls for setting following properties: ColumnsCount and ColumnsGap.

4 – The “Styles” group allows setting StyleName property.

### GroupBand Tab Set

The GroupBand tab set includes “Format” tab.



The “Format” tab for GroupBand includes following groups:

1 – The “Render” group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.

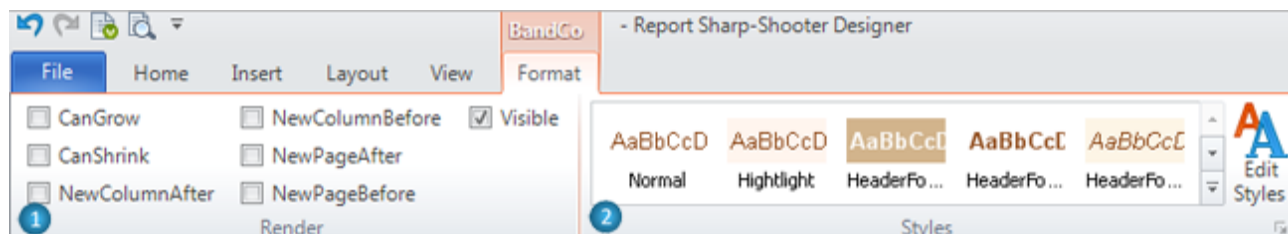
2 – The “Data Settings” group contains controls for setting following properties: GroupExpression, SortExpression and Order.

**3** – The “Columns” group contains controls for setting following properties: ColumnsCount and ColumnsGap.

**4** – The “Styles” group allows setting StyleName property.

### *BandContainer Tab Set*

The BandContainer tab set includes “Format” tab.



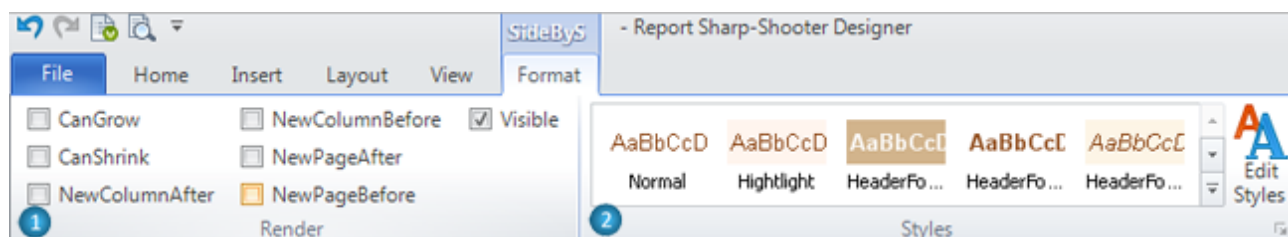
The “Format” tab for BandContainer includes following groups:

**1** – The “Render” group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.

**2** – The “Styles” group allows setting StyleName property.

### *SideBySide Tab Set*

The SideBySide tab set includes “Format” tab.



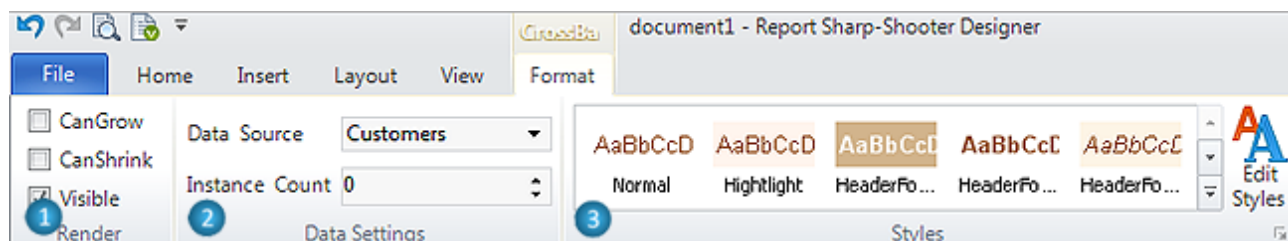
The “Format” tab for SideBySide includes following groups:

**1** – The “Render” group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.

**2** – The “Styles” group allows setting StyleName property.

### *CrossBand Tab Set*

The CrossBand tab set includes “Format” tab.



The “Format” tab for CrossBand includes following groups:

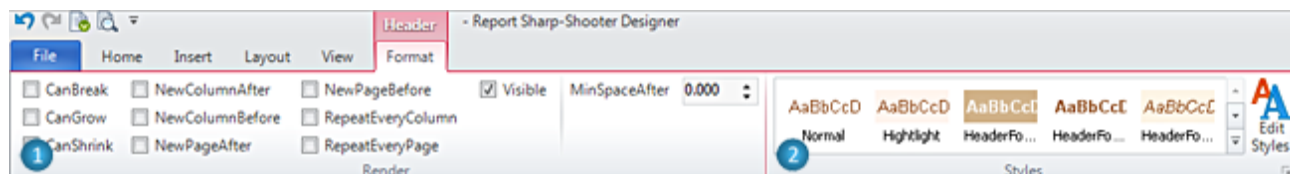
**1** – The “Render” group contains Checkboxes for setting Render properties: CanGrow, CanShrink and Visible.

**2** – The “Data Settings” group contains controls for setting following properties: DataSource and InstanceCount.

**3** – The “Styles” group allows setting StyleName property.

### Header Tab Set

The Header tab set includes “Format” tab.



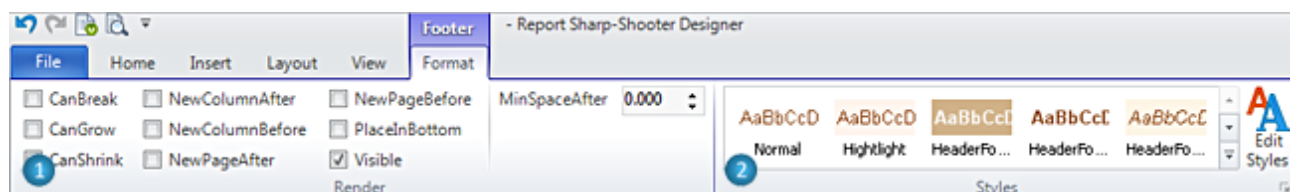
The “Format” tab for Header includes following groups:

**1** – The “Render” group contains Checkboxes for setting Render properties: CanBreak, CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore, RepeatEveryColumn, RepeatEveryPage and Visible; counter MinSpaceAfter sets MinSpaceAfter property.

**2** – The “Styles” group allows setting StyleName property.

### Footer Tab Set

The Footer tab set includes “Format” tab.



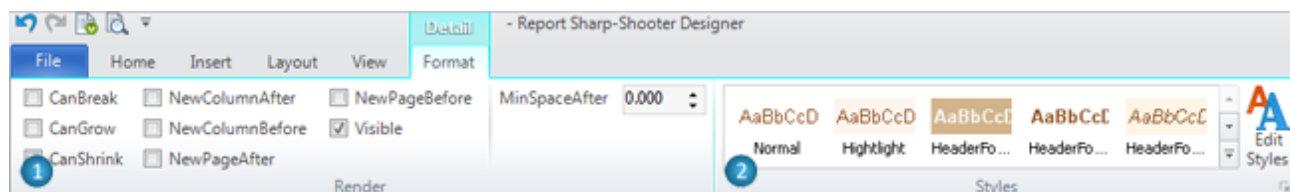
The “Format” tab for Footer includes following groups:

**1** – The “Render” group contains Checkboxes for setting Render properties: CanBreak, CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore, PlaceInBottom and Visible; counter MinSpaceAfter sets MinSpaceAfter property.

**2** – The “Styles” group allows setting StyleName property.

### Detail Tab Set

The Detail tab set includes “Format” tab.



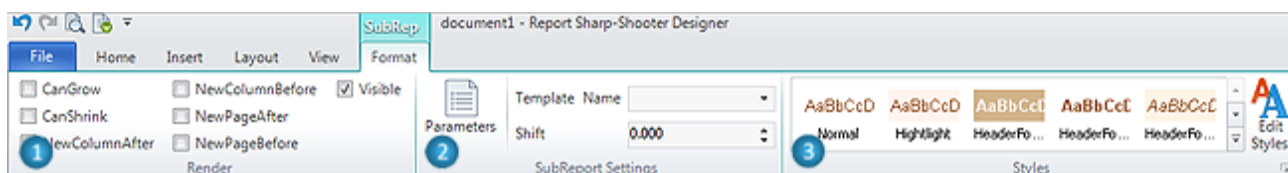
The “Format” tab for Detail includes following groups:

**1** – The “Render” group contains Checkboxes for setting Render properties: CanBreak, CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible; counter MinSpaceAfter sets MinSpaceAfter property.

**2** – The “Styles” group allows setting StyleName property.

### SubReport Tab Set

The SubReport tab set includes “Format” tab.

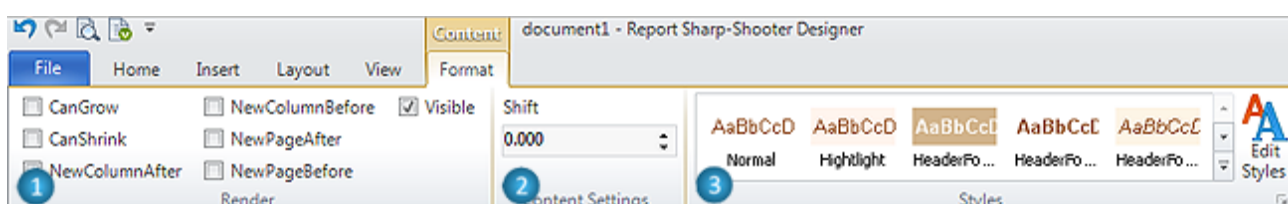


The "Format" tab for SubReport includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.
- 2 – The "SubReport Settings" group contains controls for setting following properties: Parameters, TemplateName and Shift.
- 3 – The "Styles" group allows setting StyleName property.

### Content Tab Set

The Content tab set includes "Format" tab.

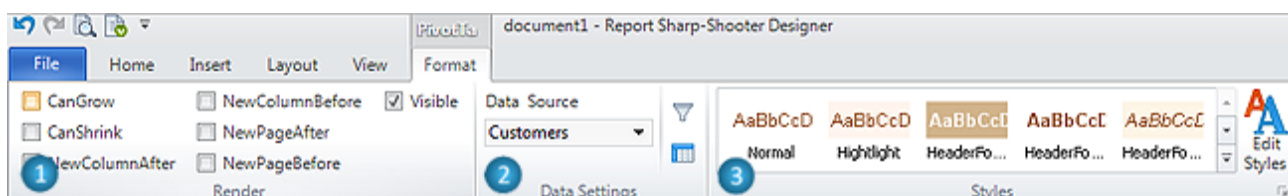


The "Format" tab for Content includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.
- 2 – The "Content Settings" group contains control for setting Shift property.
- 3 – The "Styles" group allows setting StyleName property.

### PivotTable Tab Set

The PivotTable tab set includes "Format" tab.



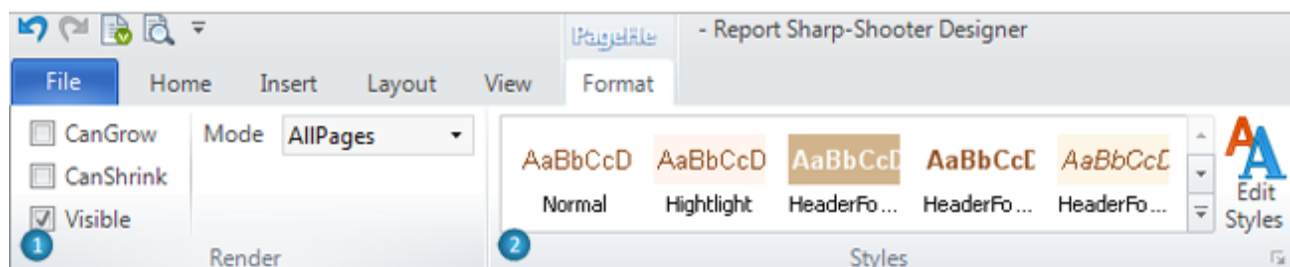
The "Format" tab for PivotTable includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, NewColumnAfter, NewColumnBefore, NewPageAfter, NewPageBefore and Visible.
- 2 – The "Data Settings" group contains controls for setting following properties: DataSource, FilterExpression and Layout.
- 3 – The "Styles" group allows setting StyleName property.



### PageHeader Tab Set

The PageHeader tab set includes "Format" tab.

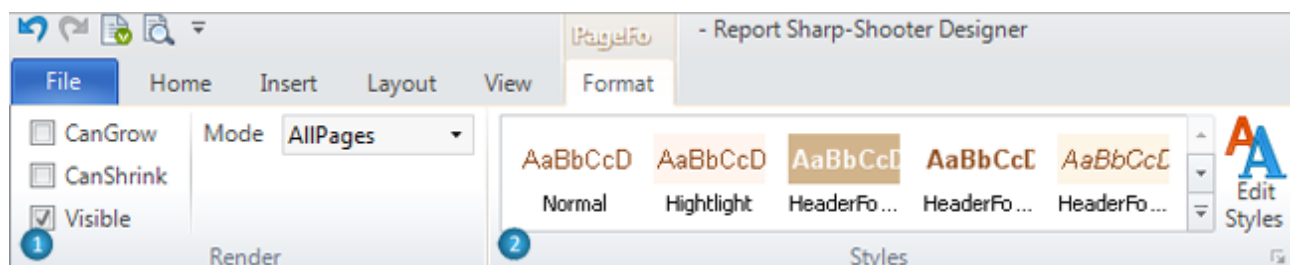


The "Format" tab for PageHeader includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, Visible and list Mode, which contains Mode property value.
- 2 – The "Styles" group allows setting StyleName property.

### PageFooter Tab Set

The PageFooter tab set includes "Format" tab.

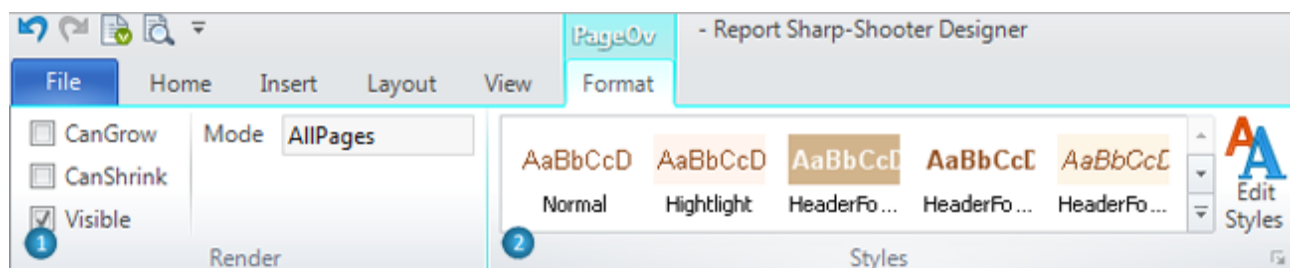


The "Format" tab for PageFooter includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, Visible and list Mode, which contains Mode property value.
- 2 – The "Styles" group allows setting StyleName property.

### PageOverlay Tab Set

The PageOverlay tab set includes "Format" tab.

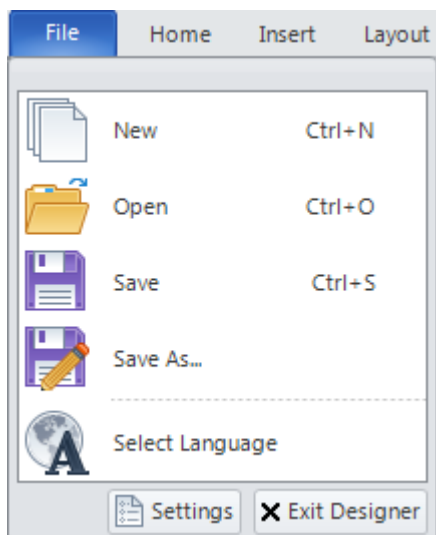


The "Format" tab for PageOverlay includes following groups:

- 1 – The "Render" group contains Checkboxes for setting Render properties: CanGrow, CanShrink, Visible and list Mode, which contains Mode property value.
- 2 – The "Styles" group allows setting StyleName property.



## Application Menu



The "New" menu item opens Wizards Gallery dialog for the creation of a new template.

The "Open" menu item opens "Open Template" window for opening template or document.

The "Save" menu item saves changes in the edited template or document. If the template or document is not saved, opens the "Save Template" window for selection of folder for saving and file name.

The "Save As..." menu item opens the "Save Template" window for selection of folder and file name to save a report or document.

The "Select Language" menu item opens the "Select Language" window for the selection of localization language.

The "Settings" button opens "Palette Settings" window for selection of Theme and Color scheme of the Designer.

The "Exit Designer" button closes Report Designer.





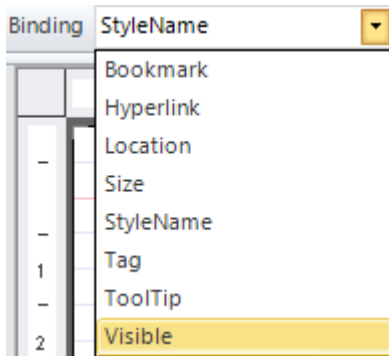
## Binding Bar

Binding Bar allows to set binding objects' properties quickly.



**1** – Name – This field sets the name of the selected object.

**2** – Binding – This is a list of dynamic properties of the selected object. It allows the selection of the property for a value setting.



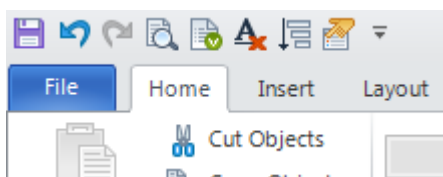
**3** – Script Editor – This button opens the Script Editor for setting the expression of the selected object.

**4** – F(x) – This field sets the expression to set dynamic property for selected object.

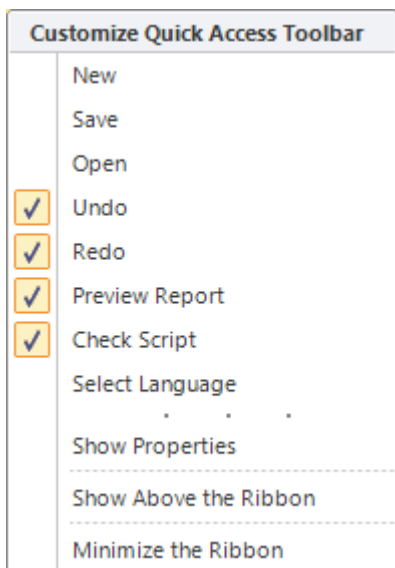
(Read more about Using Scripts)

## Quick Access Toolbar

The Quick Access Toolbar is used to provide access to frequently used commands.



Customization allows adding necessary commands.



The next command can be added to Quick Access Toolbar:

Button	Action
New Document	Create a new template
Save	Save changes in a template
Open	Load a template or document from file
Select Language	Open "Select Language" dialog
Font	Open "Font" dialog
Use Style Only	Set the style settings
Fill	Open "Fill Editor" to set Fill property
Text Fill	Open "Fill Editor" to set TextFill property
Border	Open "Border Editor"
Margins	Open "Margins Editor"
Text Format	Open "Format Editor"
Style Sheet Editor	Open "Style Sheet Editor"
Align	Open "Align" dialog
Actual Size	Set zoom to 100%
Page Width	Fit to page width
Whole Page	Fit to whole page
Properties	Open "Properties" window
Document Tree	Open "Document Tree" window
Data Source	Open "Data Source" window
Error List	Open "Error List" window
Show Grid	Switch on/off the display of a grid



Snap to Grid	Switch on/off the option of binding to a grid
Show Rulers	Switch on/off the display of a rule
Show Objects Bounds	Switch on/off the display of objects bounds
Layout Bands	Switch on/off the option of automatic layout of the bands
Template Properties	Open template properties

## Context Menu

Context menu contains most frequently used items. List of menu items depends on currently selected report objects.

### Common Actions



**1** Cut Objects – This menu item cuts the selected objects and put it on the Clipboard.

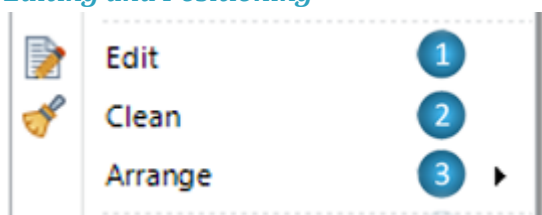
**2** Copy Objects – This menu item copies the selected objects and put it on the Clipboard.

**3** Paste – This menu item pastes the contents of the Clipboard into the selected area.

**4** Delete Objects - This menu item deletes the selected

objects.

### Editing and Positioning



**1** Edit – This menu item opens an editor:

- Text Editor for TextBox element to set TextBox.Text property;
- Formatted Text Editor for AdvancedText element to set AdvancedText.Text property;
- RichText Editor for RichText element to set RichText.RTFtext property;
- Open file dialog to choose image and to set Picture.Image property;
- Text Editor for BarCode element to set BarCode.Code property;
- SharpShooter Charts designer for ChartControl element to set ChartControl.Chart property;
- Instrument Designer for Widget element to set Widget.Instrument property;
- Source Fields for PivotTable section to set PivotTable.Layout property.

**2** Clean – This menu item cleans the content:

- TextBox.Text property for TextBox element;
- AdvancedText.Text property for AdvancedText element;
- RichText.RTFtext property for RichText element;
- Picture.Image property for Picture element;
- BarCode.Code property for BarCode element;
- ChartControl.Chart property for ChartControl;
- Widget.Instrument property for Widget element.

**3 Arrange** – This submenu contains items for managing objects order, position and size.



**3.1 Bring to Front** - This menu item brings the selected objects in front of all other objects so that no part of it is hidden behind another object in the current container.

**3.2 Move Forward** - This menu item brings the selected object forward so that it is hidden by fewer objects that are in front of it in the current container.

**3.3 Move Back** - This menu item sends the selected objects backward so that it is hidden by the objects that are in front of it in the current container.

**3.4 Send to Back** - This menu item sends the selected objects behind all other objects in the current container.

The following items are available only if multiple objects are selected:

**3.5 Align Left** – This menu item aligns lefts of the selected objects by the left border of the most left object.

**3.6 Align Center** - This menu item aligns centers of the selected objects by the center between the most left and the most right objects.

**3.7 Align Right** - This menu item aligns rights of the selected objects by the right border of the most right object.

**3.8 Align Top** - This menu item aligns tops of the selected objects by the top border of the most top

object.

**3.9 Align Middle** - This menu item aligns middles of the selected objects by the middle between the most top and the most bottom objects.

**3.10 Align Bottom** - This menu item aligns bottoms of the selected objects by the bottom border of the most bottom object.

**3.11 Distribute Horizontally** – This menu item distributes selected objects horizontally making horizontal spaces between them equal to each other.

**3.12 Distribute Vertically** – This menu item distributes selected objects vertically making vertical spaces between them equal to each other.

**3.13 Same Height as <main selected object\*>** - This menu item sets height of every selected object as the main selected object's height.

**3.14 Same Width as <main selected object>** - This menu item sets width of every selected object as the main selected object's width.

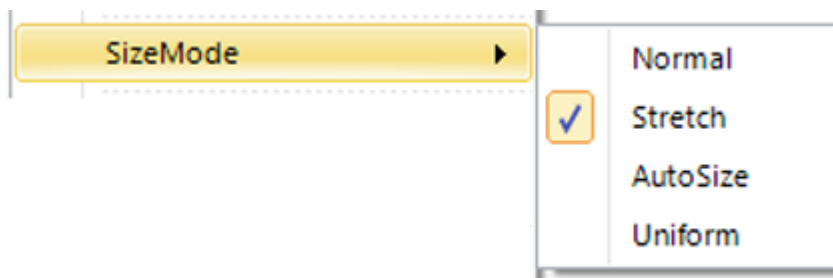
**3.15 Same Size as <main selected object>** - This menu item sets size of every selected object as the main selected object's size.

\* Main selected object – object, which was last selected at current selected objects. This object is marked in a darker blue.

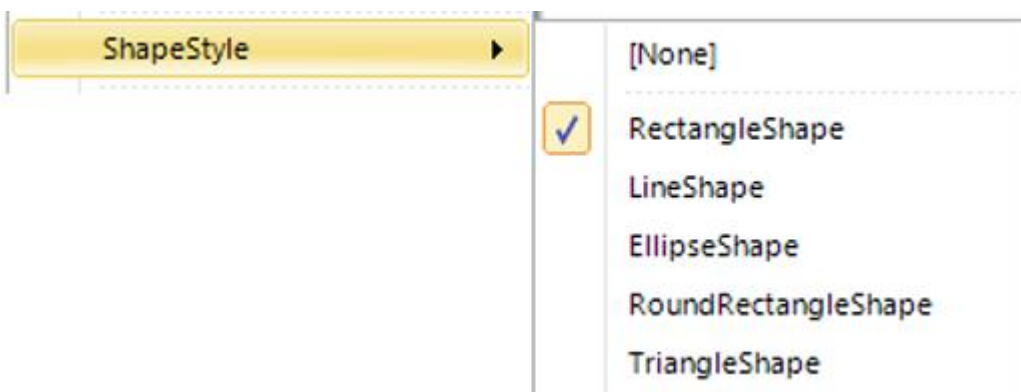
### Element-dependent Menu Items

Here are menu items used for setting unique objects' properties.

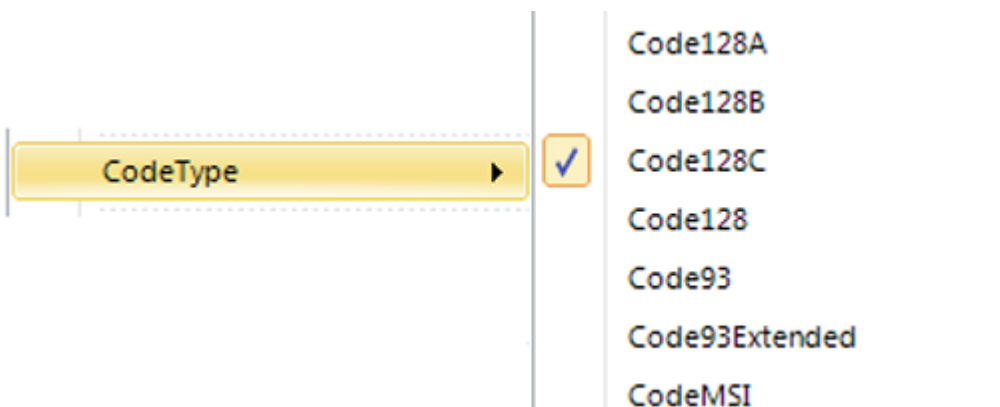
SizeMode – This is for setting Picture.SizeMode property; the submenu under it contains the list of SizeMode property values.



ShapeStyle – This menu item is for setting Shape.ShapeStyle property; the submenu under it contains the list of ShapeStyle property values.



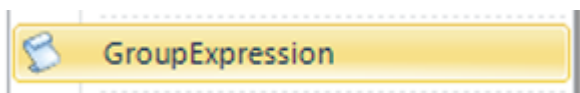
CodeType – This menu item is for setting BarCode.CodeType property; the submenu under it contains the list of CodeType property values.



DataSource – This menu item is for setting DataSource property of DataBand, PivotTable and CrossBand sections; the submenu under it contains the list of data sources of the template.



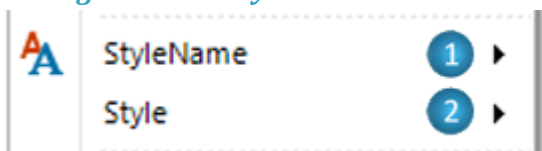
GroupExpression – This menu item opens Script Editor for setting GroupBand.GroupExpression property.



SubReport – This menu item is for setting SubReport.TemplateName property; the submenu under it contains the list of Report Names contained in the current ReportManager.



### Editing Elements Styles



**1** StyleName – This submenu is for setting StyleName property of the objects; contains the list of styles defined in the template.

**2** Style – This submenu contains the following items:



**2.1** Use Style Only – This menu item clears all formatting settings (e.g. font, border, etc.) of the element; appearance of the element is defined by style currently applied to.

**2.2** Font – This menu item opens Font dialog for setting Font property;

**2.3** TextFormat – This menu item opens Format dialog for setting TextFormat property;

**2.4** Border – This menu item opens Border Editor for setting Border property;

**2.5** Fill - This menu item opens Fill Editor for setting Fill property;

**2.6** TextFill - This menu item opens Fill Editor for setting TextFill property;

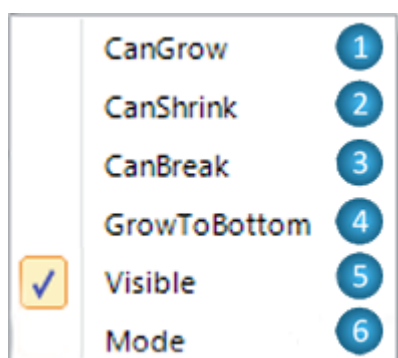
**2.7** Align – This menu item is for setting TextAlign or ImageAlign properties; submenu under it contains align options;

**2.8** Margins - This menu item opens Margins Editor for setting Margins property;

**2.9** Angle – This menu item is for setting Angle property; contains the list of allowed property values;

### Render Submenu

This submenu provides ability to set elements' properties that affect their rendering.



**1** CanGrow – This menu item sets CanGrow property to True or False.

**2** CanShrink – This menu item sets CanShrink property to True or False.



- 3 CanBreak – This menu item sets CanBreak property to True or False.
- 4 GrowToBottom – This menu item sets GrowToBottom property to True or False.
- 5 Visible – This menu item sets Visible property to True or False.
- 6 Mode – This menu item sets Mode property; submenu under it contains the list of allowed property values.

### Misc Items

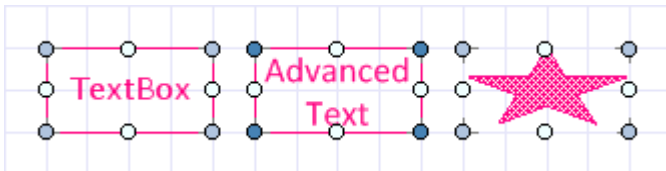


Show Properties – The menu item shows Property Grid with the selected objects' properties.

### Group selection

When multiple objects are selected, context menu contains items which are common for all selected objects. Commands are applied to each selected object.

AdvancedText element is selected objects main at the following picture:



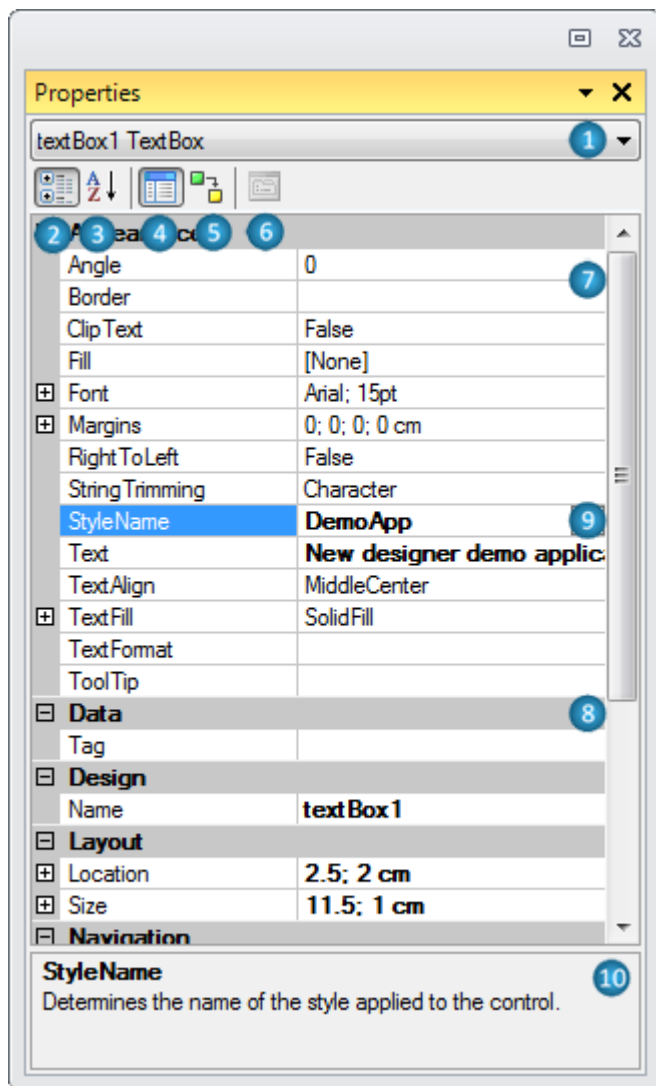


## Docking

The new Report Designer with ribbon interface supports dockable windows. Properties, DataSource, Document Tree and Error List windows are dockable. It means that windows can be attached, or docked, to any side of the designer window, or it can be detached, or floated, in its own mini-frame window.

## Properties

Property Grid is intended to display and edit properties of the selected element.



**1** – Object List – This is a list of template objects which displays all template objects. Property Grid displays the object properties selected in Object List. The first word is an object name, the second – an object type.

**2** – Categorized – This button switches on the mode of properties display by categories.

**3** – Alphabetical – This button switches on the mode of properties display in alphabetic order.

**4** – Properties – This button switches on the mode of static properties display.

**5** – Bindings – This button switches on the mode of dynamic properties display.

**6** – Property Pages. This button is not used.

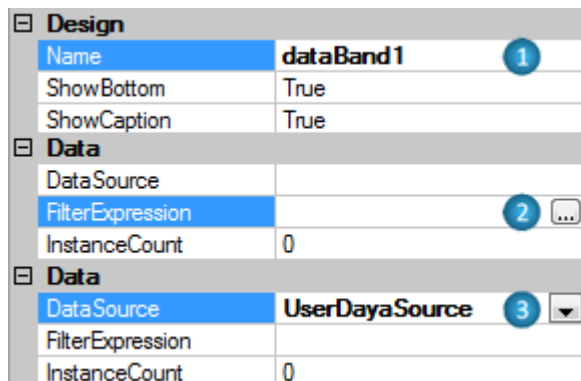
**7** – Property Grid – Object property grid.

**8** – Group of properties.

**9** – Properties: name and value.

**10** – Description – Description of the selected property.

Different methods are used for properties setting:



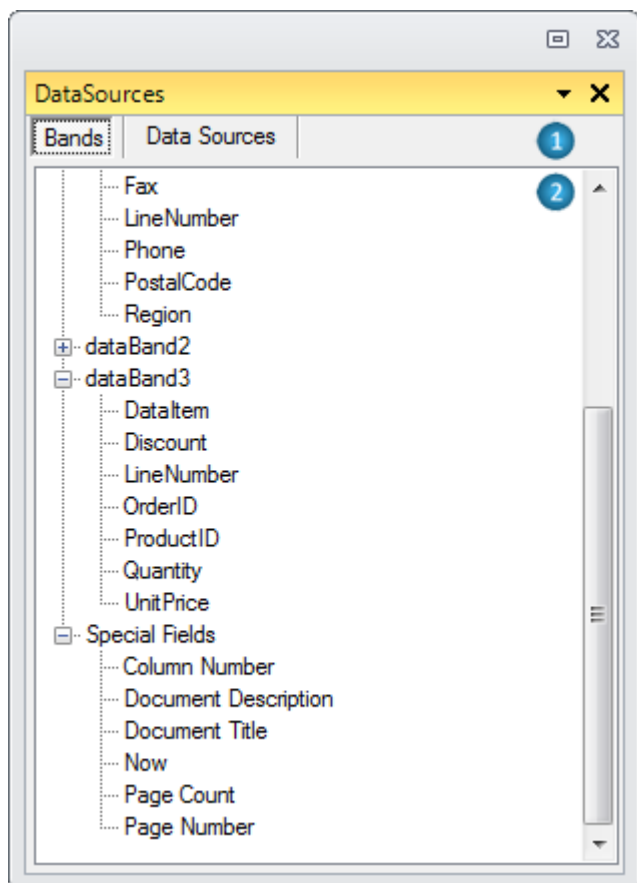
**1** – Field for input of text and numeric values.

**2** – This button opens the property editor.

**3** – List of values.



### Data Sources



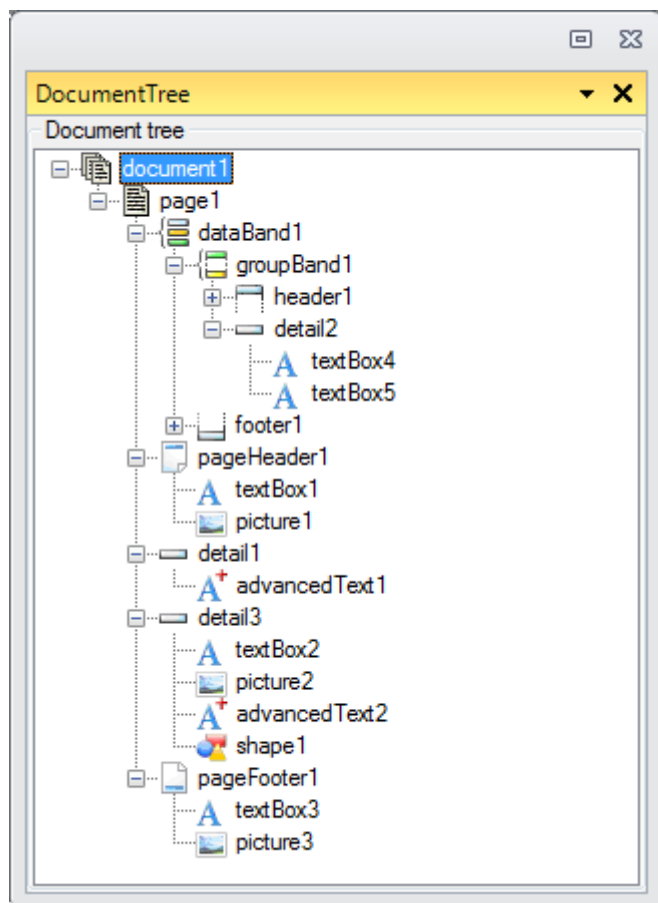
**1** – Switching between Bands and Data Sources parts. Data from report bands and special fields are displayed in Band part. Data from report data sources are displayed in Data Sources part.

**2** –The tree of fields from data source allows the creation of TextBox objects for information output from data source by dragging the fields.



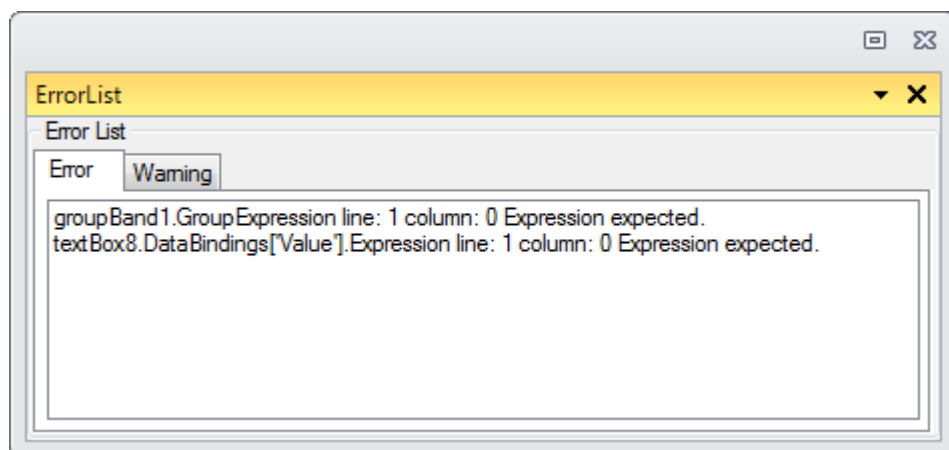
### Document Tree

The document tree represents a set of objects ordered according to the report structure. It allows the selection of the object for properties editing, the deletion of objects, changing the template structure.



### Error List

The list of scripts errors includes band name where the error was made and also a description of an error. It is possible to open the Script Editor for expression or script editing by double click with a left mouse button.



(More information about errors in scripts)

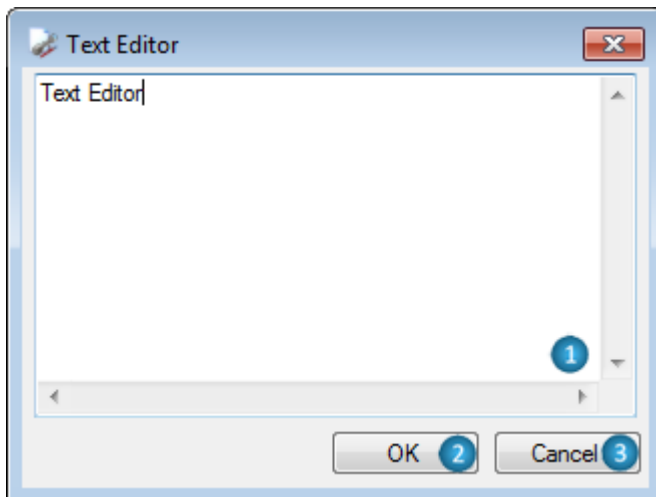


## Editors

### Text Editor

This editor is for input of text values. It allows the input of simple text.

The editor is opened by double click on the TextBox element area, hotkeys Enter or Ctrl + Enter, when TextBox element is selected, or by button, which is located near the property on Property Grid.



1 – Area for text input.

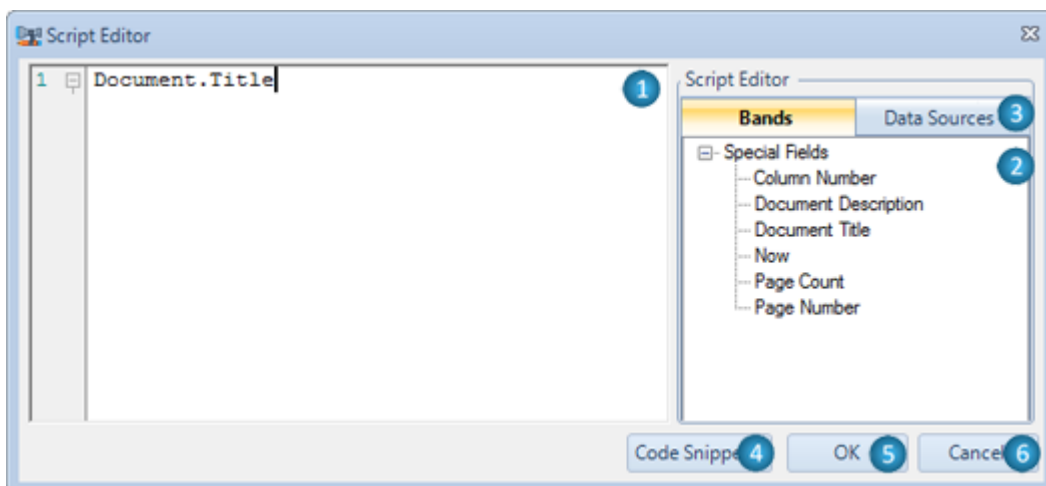
2 – OK – This button applies changes, closes the editor.

3 – Cancel – This button cancels changes, closes the editor.

### Script Editor

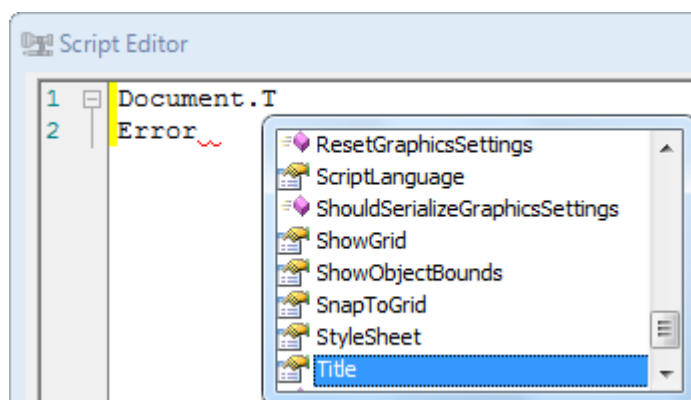
This editor is for input of scripts and expression of various destinations.

The editor is opened by the button, which is located near the property on the Property Grid or by the "Script Editor" button on the Binding bar.



1 – Area for script input.

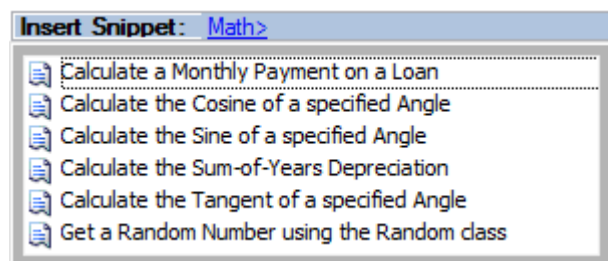
Script Editor provides syntax highlighting, changes selection and code auto-completing.



**2** – Fields' tree displays the fields from the data source (the Data Sources part), from the bands and displays special fields (the Bands part). The quick input of expressions can be executed by dragging the fields from the fields' tree to the area for script input or by double click on the field in the tree.

**3** – Switching between Bands and Data Sources parts.

**4** – Code Snippets – This button opens a list for templates selection to insert the code fragment.



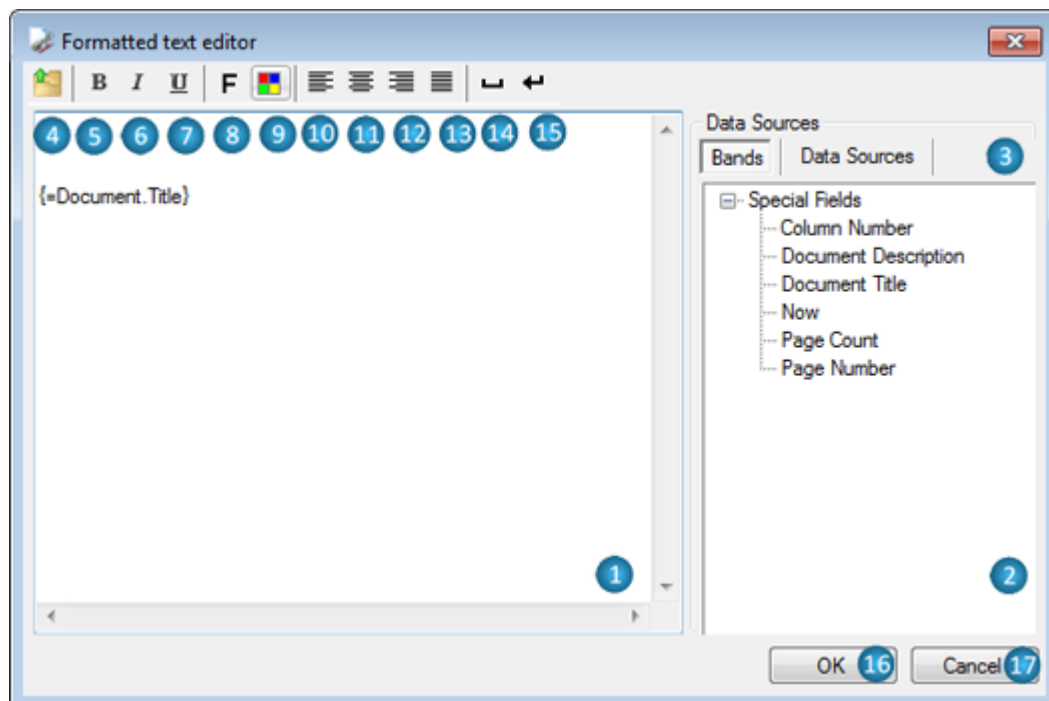
**5** – OK – This button applies changes, closes the editor.

**6** – Cancel – This button cancels the made changes, closes the editor.

### Formatted Text Editor

This editor is for input of formatted text in the form of HTML, similar markup language and for RTF-text loading.

The editor is opened by double click on the AdvancedText element area, by hotkeys Enter or Ctrl + Enter, when AdvancedText element is selected, or by the button, which is located near the Text property on Property Grid for AdvancedText element.



**1** – Area for text input.

**2** – The fields' tree displays fields from the data sources (the Data Sources part), from bands and displays special fields (the Bands part). The quick input of expressions can be executed by dragging the fields from the fields' tree to the area for script input or by double click on the field in the tree.

**3** – Switching between Bands and Data Sources parts.

**4** – Open RTF Document – This button opens the Open dialog for the selection of RTF file.

**5** – Bold – This button inserts `<b></b>` tags into the text for setting the bold font of the selected text.

**6** – Italic – This button inserts `<i></i>` tags into the text for setting the italic font of the selected text.

**7** – Underline – This button inserts `<u></u>` tags into the text for setting the underline font of the selected text.

**8** – Font – This button opens the Font editor for setting the style and font of the selected text, inserts `<font=></font>` tags into text.

**9** – Color – This button opens the Color editor for setting the color of the selected text, inserts `<font color=></font>` tags into text.

**10** – Left Align – This button inserts `<p></p>` tags into text for setting the left alignment of selected text.

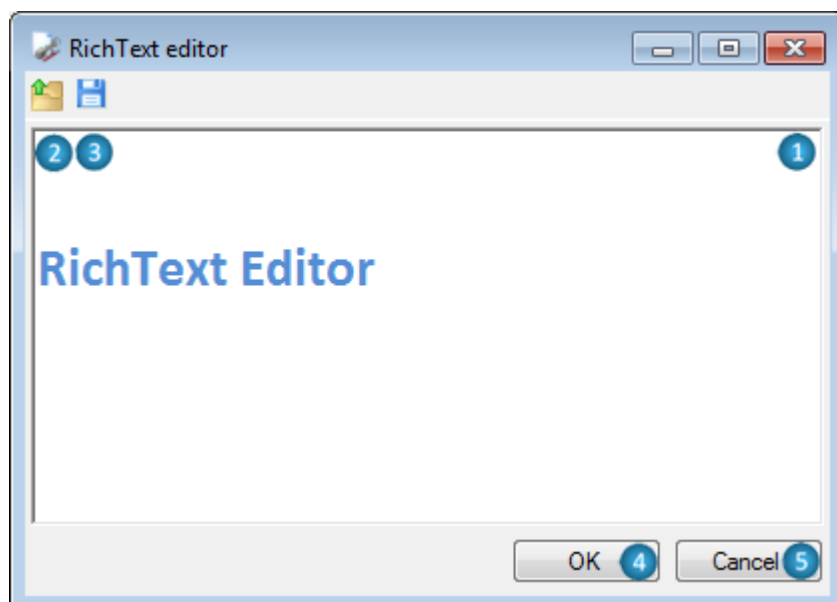


- 11** – Center Align – This button inserts `<p align="center"></p>` tags into text for setting the center alignment of selected text.
- 12** – Right Align – This button inserts `<p align="right"></p>` tags into text for setting the right alignment of selected text.
- 13** – Justify – This button inserts `<p align="justify"></p>` tags into text for setting the justify alignment of selected text.
- 14** – Insert Space – This button inserts `&nbsp;` tag into text to display a space.
- 15** – Insert Line Break – This button inserts `<br/>` tag into text for text breaking into strings.
- 16** – OK – This button applies changes, closes the editor.
- 17** – Cancel – This button cancels changes, closes the editor.

### *RichText Editor*

The editor for loading and editing RTF text.

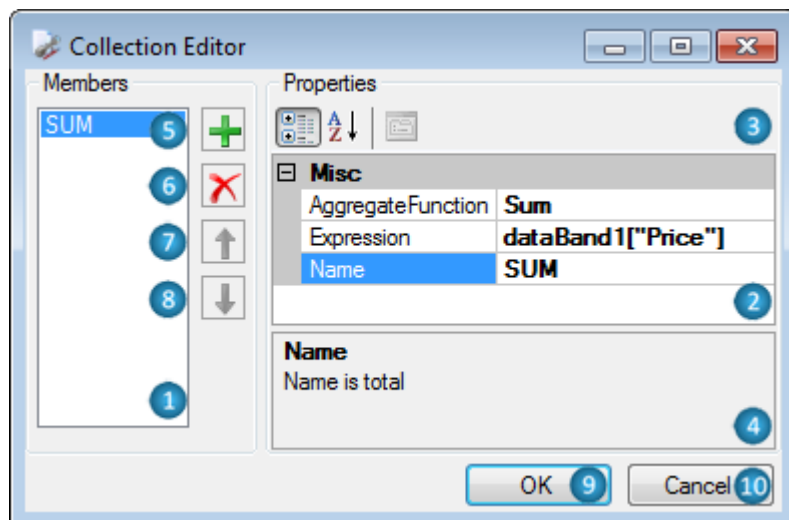
The editor is opened by double click on the RichText element area, by hotkeys Enter or Ctrl + Enter, when RichText element is selected, or by the button, which is located near the RTFText property on Property Grid for RichText element.



- 1** – The area for the text input.
- 2** – Open File – This button opens the Open dialog for selecting the RTF file.
- 3** – Save to File – This button opens the Save As dialog for text saving into the RTF file.
- 4** – OK – This button applies changes, closes the editor.
- 5** – Cancel – This button cancels the changes, closes the editor.

### Collection Editor

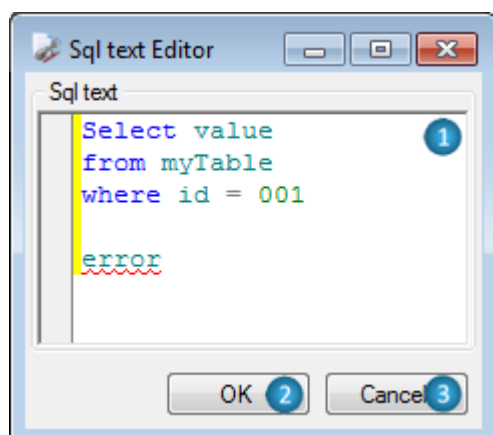
The collection editor is intended for input of members into collection and setting their properties. The editor is opened with the button, which is located near the property on the Property Grid.



- 1 – A list of collection members. The Name property is displaying in the list.
- 2 – Property Grid –The table of collection members’ properties. It allows the properties editing.
- 3 – Switching the properties display mode in Property Grid.
- 4 – This area displays the description of the selected text.
- 5 – This button adds a new member to the collection.
- 6 – This button deletes the selected member from the collection.
- 7 – This button moves the member upwards through the list.
- 8 – This button moves the member downwards through the list.
- 9 – OK – This button applies changes, closes the editor.
- 10 – Cancel – This button cancels the changes, closes the editor.

### SQL Text Editor

This editor is for SQL commands input. The editor is opened with the button from the Document Data Sources editor for data source setting. This button is located near the SelectQuery property on Property Grid.



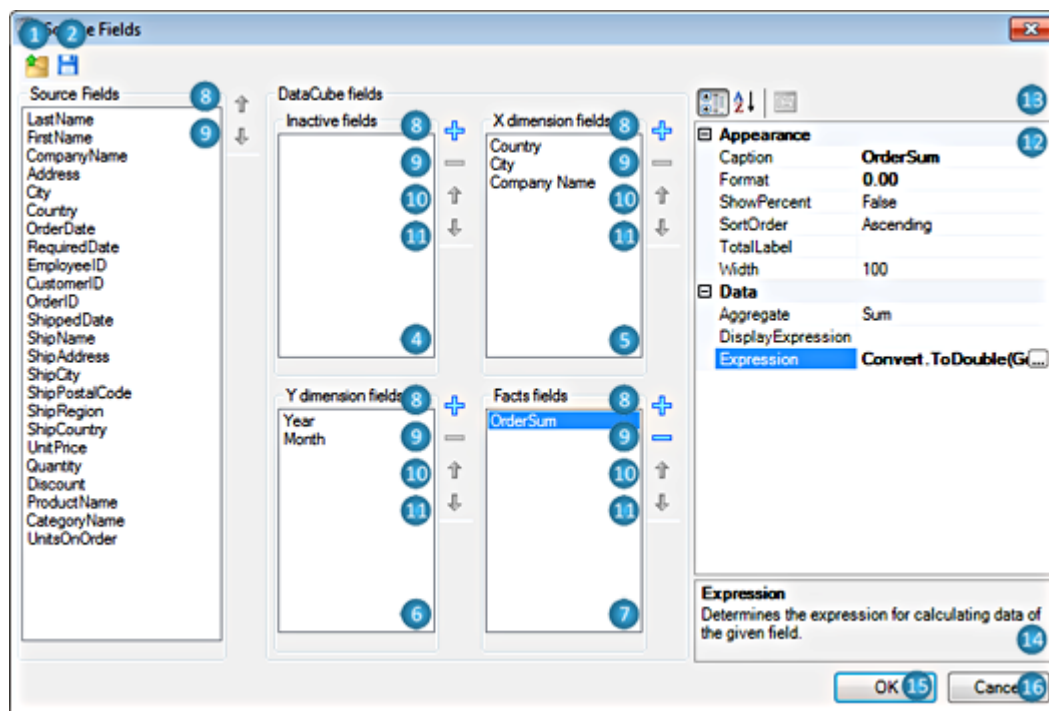




- 1 – The area for the text output. The SQL scripts commands editor provides the highlighting of syntax and changes selection.
- 2 – OK – This button applies changes, closes the editor.
- 3 – Cancel – This button cancels the changes, closes the editor.

### Source Fields

The Pivot Table data editor allows the setting of the format and layout of data. The editor is opened with the button, which is located near the Layout property of the PivotTable or by the Configure Layout button on the Pivot Table Wizard form.



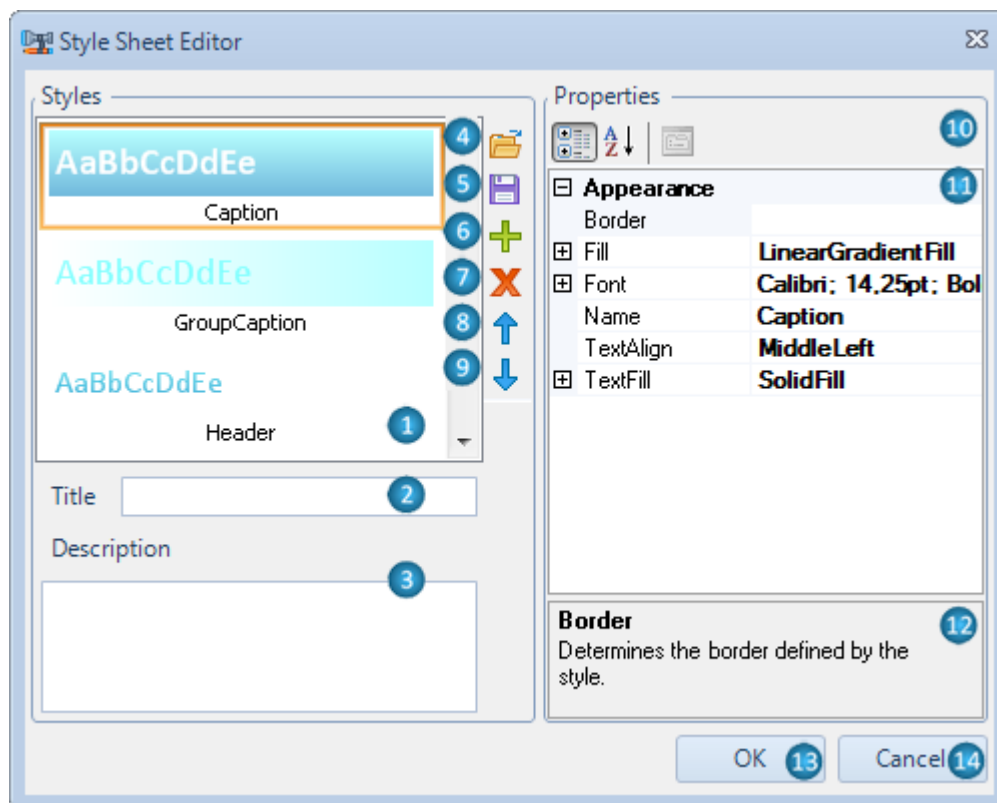
- 1 – Open – This button opens the “Open” window for selection of the Data cube layout file (.dcl) for loading the data settings.
- 2 – Save – This button opens the “Save As” window for saving the data settings.
- 3 – Source fields – The list of fields from the data source.
- 4 – Inactive fields – The list of fields for inactive fields.
- 5 – X dimension fields – The list of fields for horizontal headline of the table.
- 6 – Y dimension fields – The list of fields for vertical headline of the table.
- 7 – Facts fields – The list of fields for table data.
- 8 – New field – This button adds a new empty field to the list.
- 9 – Remove – This button deletes the selected field from the list.
- 10 – Up – This button moves the field upwards in the list.
- 11 – Down – This button moves the field downwards in the list.
- 12 – Property Grid – The field properties grid which allows the editing of properties.
- 13 – Switching the properties display mode in Property Grid.
- 14 – Description – This area displays the description of the selected property.

**15** – OK – This button applies changes, closes the editor.

**16** – Cancel – This button cancels the changes, closes the editor.

### Style Sheet Editor

The styles editor allows the creation and editing of the template styles. The editor is opened by the "Edit Style" button on the "Home" and "Format" ribbon tabs.



**1** – Style – List of styles.

**2** – Title – Name of styles set.

**3** – Description – This area displays the description of styles set.

**4** – Open – This button opens the Load Style Sheet dialog for selection of the StyleSheet file (.rss) for styles loading.

**5** – Save – This button opens the Save Style Sheet dialog for saving the styles set.

**6** – Add Style – This button adds a new style to the set.

**7** – Remove Style – This button deletes the selected style from the set.

**8** – Move Up Style – This button moves the selected style upwards in the list.

**9** – Move Down Style – This button moves the selected style downwards through the list.

**10** – Switching the properties display mode in Property Grid.

**11** – Property Grid – The style property grid which allows the editing of properties: Border, Fill, Font, Name, TextAlign, TextFill.

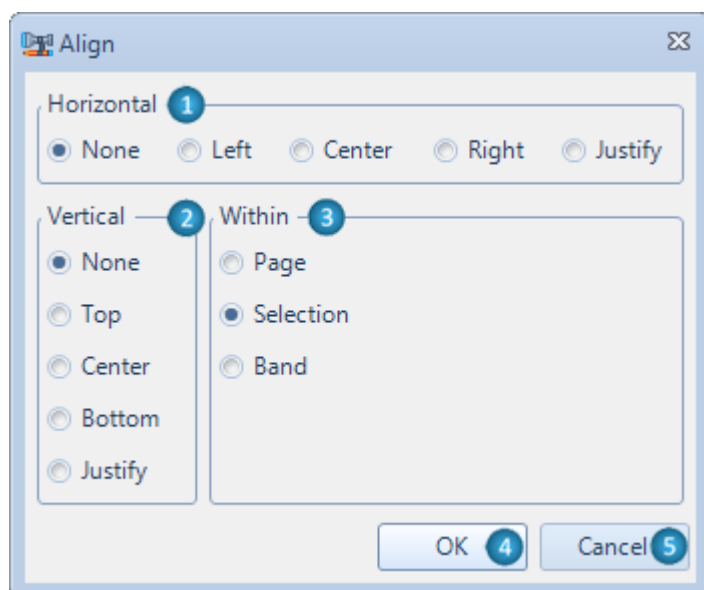
**12** – Description – This area displays the description of the selected property.

**13** – OK – This button applies changes, closes the editor.

**14** – Cancel – This button cancels the changes, closes the editor.

### *Align Editor*

The editor allows the setting of the elements alignment in a template. The editor is opened with the “Align” button on the “Layout” ribbon tab.



**1** – Horizontal – The settings of horizontal alignment allow the changing of location and size of the selected objects.

None – Without alignment;

Left – Alignment by the left border of the most left object (sets the Location property);

Center – Alignment by center between the most right and the most left objects (sets the Location property);

Right – Alignment by the right border of the most right object (sets the Location property);

Justify – Alignment by width on the gap between the right and left objects (sets the Size and Location properties).

**2** – Vertical – The settings of vertical alignment allows the changing of location and size of the selected elements:

None – Without alignment;

Top – Alignment by the top border of the most top object (sets the Location property);

Center – Alignment by center between the most top and the most bottom object (sets the Location property);

Bottom – Alignment by the bottom border of the most bottom object (sets the Location property);

Justify – Alignment by height on the gap between the top and bottom objects (sets the Size and Location properties).

**3** – Within – The setting of center alignment allow the changing of location of the selected elements:

Page – Alignment by center of the page on which the selected objects are located (sets the Location property);



Selection – Alignment by center of the area occupied by the selected objects (sets the Location property);

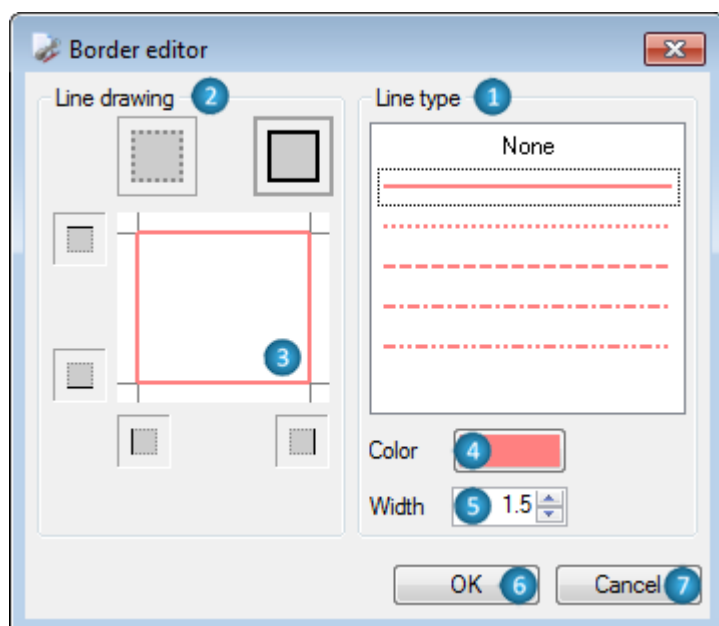
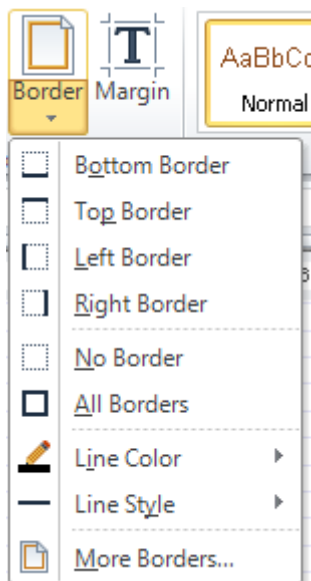
Band – Alignment by center of the band-container in which the selected objects are located (sets the Location property);

**4** – Apply – This button applies changes.

**5** – Close – This button closes the editor.

### Border Editor

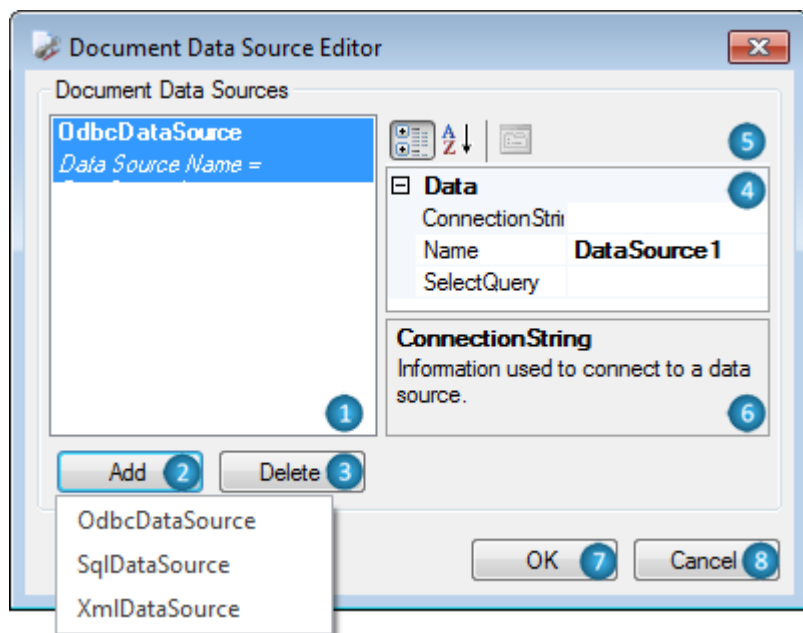
This editor is for setting elements' border. The editor is opened with the button, which is located near the Border property on Property Grid or by the "More Borders..." menu item (the "Border" button on the "Home" and some contextual tabs).



- 1 – Line Type – A list of line types which allows the selection of the line type for the border.
- 2 – Line Drawing – The area for selection of borders lines location.
- 3 – The display area of set borders.
- 4 – Color – This button opens the Color editor for selection of the border color and displays current color.
- 5 – Width – Counter sets the width of the border line.
- 6 – OK – This button applies changes, closes the editor.
- 7 – Cancel – This button cancels changes, closes the editor.

## Document Data Sources

The editor of the embed data sources collection. It allows the setting of data sources for the template (the Document.DataSources property). The editor is opened with the button, which is located near the DataSource property of the Document object on Property.



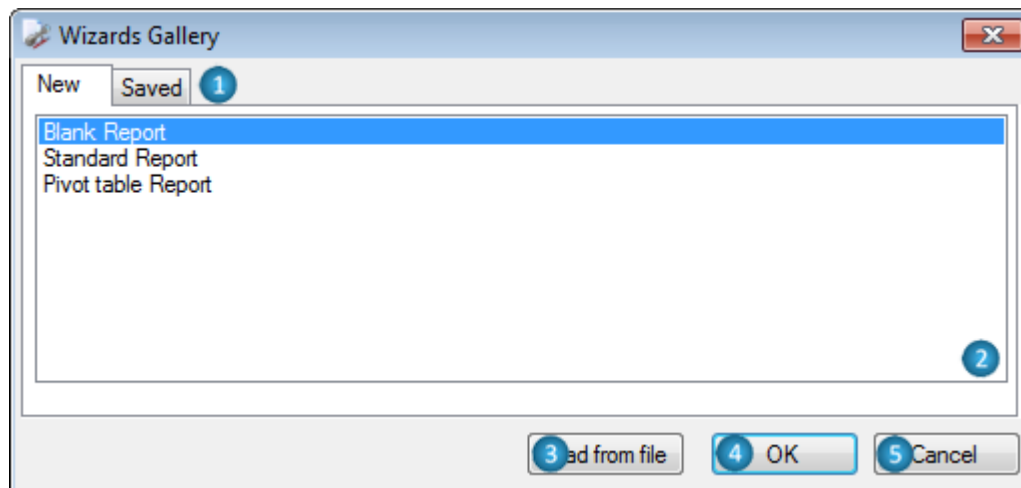
- 1 – Document Data Sources – A list of data sources for the document.
- 2 – Add – This button opens a list of data source types for addition to a template: SqlDataSource, OdbcDataSource, and XmlDataSource.
- 3 – Delete – This button deletes the selected data source.
- 4 – Property Grid – Data source property grid which allows the property editing.
- 5 – Switching the property display mode in Property Grid.
- 6 – Description – This area displays the description of the selected text.
- 7 – OK – This button applies changes, closes the editor.
- 8 – Cancel – This button cancels changes, closes the editor.

## Wizards

Wizards are used for creation of simple reports on the fly.

### Wizards Gallery

It allows the selection of a method that will be used to create a simple report. It is invoked with "New" item of Application menu command or with the key combination (Ctrl + N).



**1** –Switching between "New" and "Saved" tabs.

**2** –A list of methods for a new report creation:

- Blank Report – Creates a new empty report;
- Standard Report –Invokes Standard Wizard;
- Pivot Table Report – Invokes Pivot Table Wizard.

**3** – Load from file – This button invokes the "Open Wizard File" dialog window to open the Wizard file.

**4** – OK – This button creates a new empty template or opens Wizard depending on the selected parameters.

**5** – Cancel – This button closes Wizard Gallery.

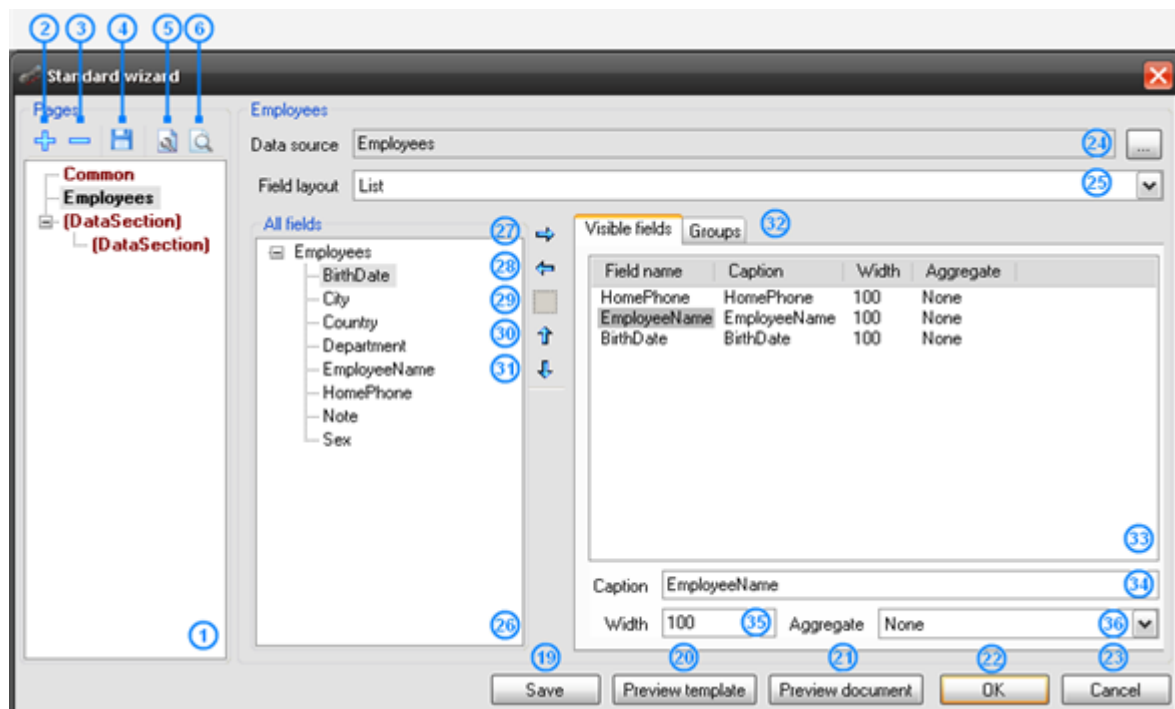
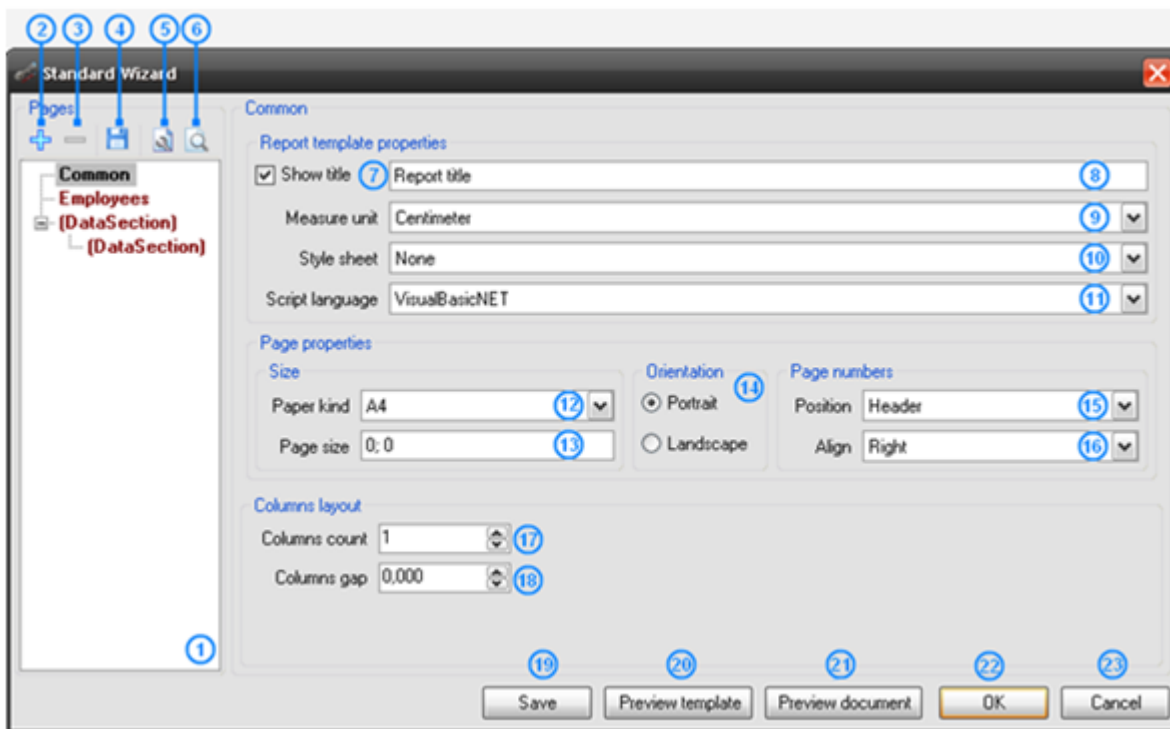
**6** – Browse – This button invokes the Browse for Folder dialog window to select the folder with saved Wizard files.

7 – Location – This field displays/sets the address of the folder with saved Wizard files.

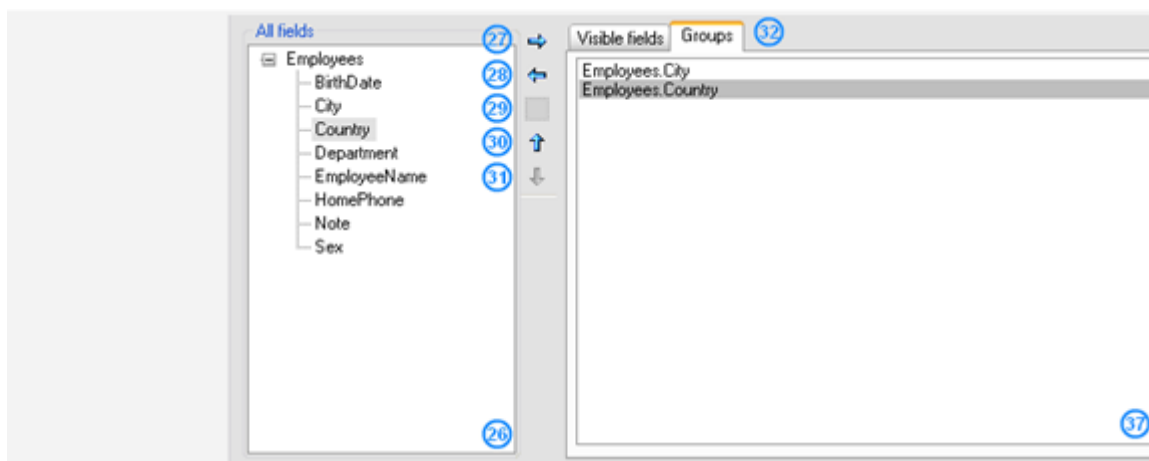
8 – A list of Wizard files which are located in the folder specified in the Location field.

### Standard Wizard

It allows the creation of standard reports which include lists and groups. It is invoked from the Wizards Gallery dialog.







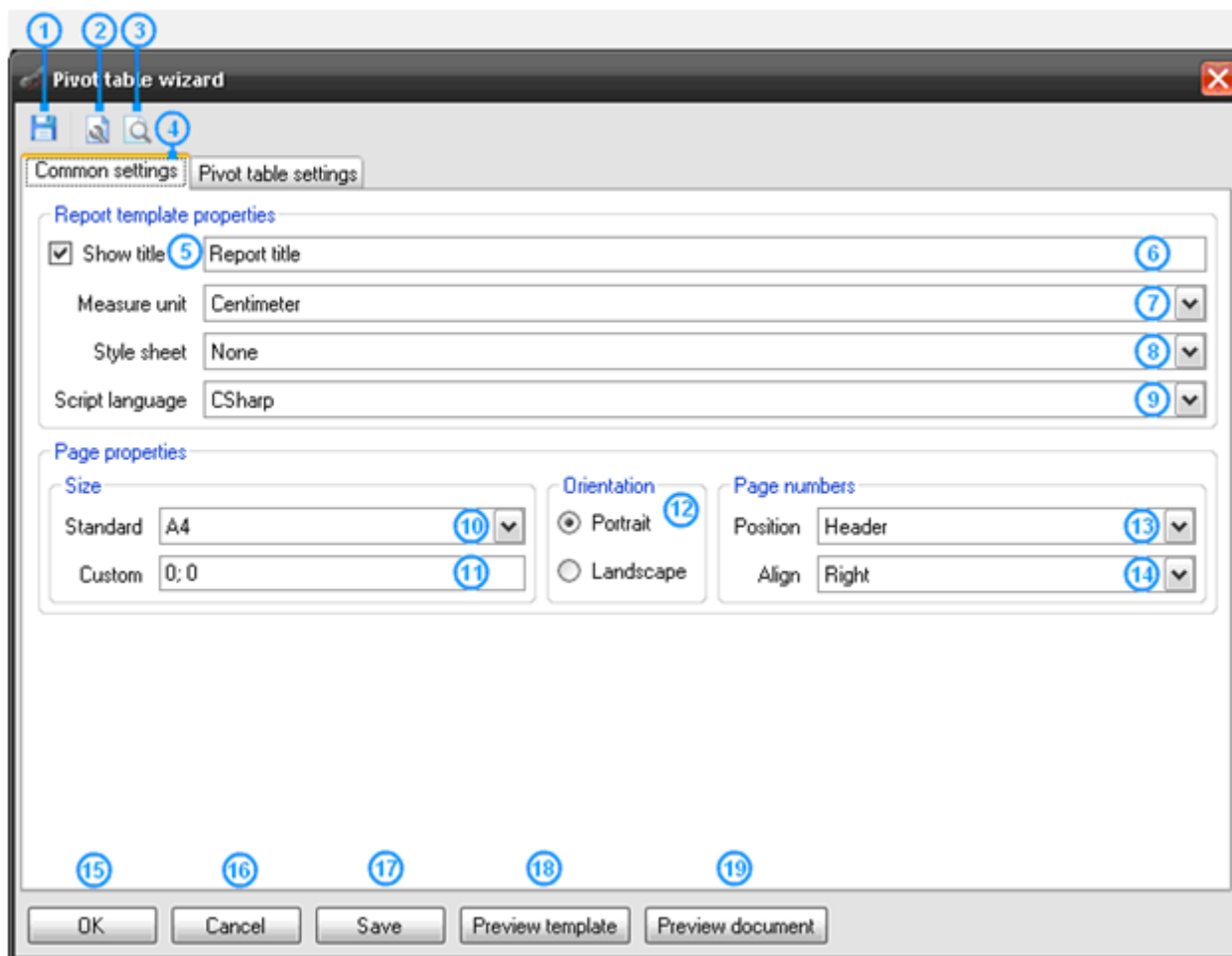
- 1** – The data tree displays a report common settings part and added data sections. It allows the switching of the settings modes.
- 2** – Add Data – This button adds a data section.
- 3** – Remove Data – This button deletes a data section.
- 4, 19** – Save Wizard – These buttons invoke the Save Wizard File dialog window for selection of the folder and file name to save Wizard settings.
- 5, 20** – Preview Template – These buttons create the template with set parameters, open it in Report Designer.
- 6, 21** – Preview Document – These button launches generation of a report with set parameters, open final document in Report Viewer.
- 7** – Show Title – Check for switching on/off report title in the report.
- 8** – Report Title – This field sets report name (sets the Document.Title property).
- 9** – Measure Unit – A list of measure units which allows the selection of measure unit for the template (sets the Document.MetricUnit).
- 10** – Style Sheet – A list of styles which allows the selection of style for the report (sets the Document.StyleSheet property).
- 11** – Script Language – A list of supported programming languages which allows the selection of language for scripts writing in the report (sets the Document.ScriptLanguage property).
- 12** – Standard – A list of page formats which allows the selection of the page format for the report (sets the Page.PaperKind property).
- 13** – Custom – This field sets the page size for the report (sets the Page.CustomSize property).
- 14** – Orientation – Radio buttons set the page orientation for the report (sets the Page.Orientation property).
- 15** – Position – A list of values of headlines positions which allows the selection of the position in the template for headlines display with page numbers (top, bottom headlines).
- 16** – Align – A list of alignment values allows the selection of number text alignment for display in headlines.
- 17** – Columns Count – The counter which sets the columns number in the list in the report (sets the DataBand.ColumnsCount property).

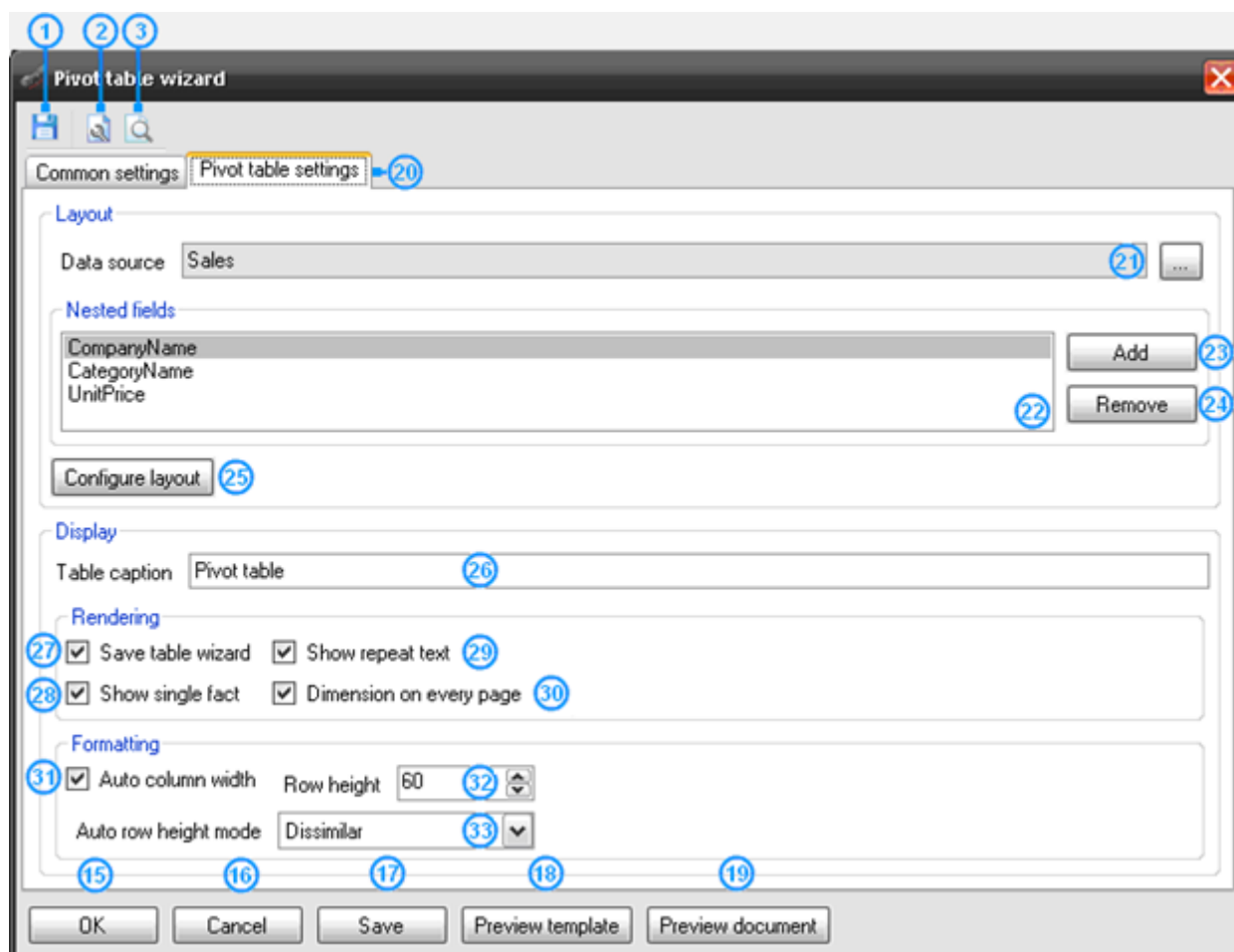


- 18** – Columns Gap – The counter which sets the gap between columns in the list in the report (sets the DataBand.ColumnsCount property).
- 22** – OK – This button shut downs Wizard work, creates a template with selected parameters, and opens it with Report Designer.
- 23** – Cancel – This buttons completes Wizard work.
- 24** – Data Source – This button opens the data sources tree for selection of a data source. The field displays the name of the selected data source.
- 25** – Field Layout – This list allows the selection of data display method.
- 26** – All Fields – Data sources fields' tree.
- 27** – Move Right – This button adds the selected field from data sources tree to the Groups list or the Visible Fields table. At this, the values of the Caption, Width, and Aggregate properties are set by default.
- 28** – Move Left – This button deletes the selected field from the Visible Fields table or the Groups list.
- 29** – Add Blank – This button adds a field with empty properties to the Visible Fields table.
- 30** – Move Up – This button moves the selected field upwards in the Visible Fields Table or the Group list.
- 31** – Move Down – This button moves the selected field downwards in the Visible Fields table or the Group list.
- 32** – Switching between the Visible Fields table and the Groups list.
- 33** – Visible Fields – This is a table of fields for report creation. It displays the added fields and their properties. The fields can be added from the data source or it can be new fields. The FieldName property displays the field name. The Caption property displays the title of the visible field. The Width property sets the column width. The Aggregates property sets the aggregation functions.
- 34** – Caption – This field sets the Caption property for record from the Visible Fields table. It sets the Textbox.Text property for the element which sets the title for field output in the report.
- 35** – Width – This field sets the Width property for record from the Visible Fields table, defines the list column width in percent.
- 36** – Aggregate – This is a list of aggregation functions which allows the selection of the function for calculation of aggregate value by field. The value will be output at the end of the list.
- 37** – Groups – This is a list of fields from data source for the list grouping.

### Pivot Table Wizard

Pivot Table Wizard allows the creation of reports which include pivot table. It invokes from Wizards Gallery dialog.





**1, 17** – Save Wizard – These buttons invoke Save As dialog window for selection of the folder and file name for saving Wizard settings.

**2, 18** – Preview Template – These buttons create a template with selected parameters, open it in Report Designer.

**3, 19** – Preview Document – These buttons launch report generation with selected parameters, open the result document in Report Viewer.

**4** – Common Settings – Report common settings tab.

**5** – Show Title – Check for switching on/off report title in the report.

**6** – Report Title – This field sets report name (sets the Document.Title property).

**7** – Measure Unit – A list of measure units which allows the selection of measure unit for the template (sets the Document.MetricUnit).

**8** – Style Sheet – A list of styles which allows the selection of style for the report (sets the Document.StyleSheet property).

**9** – Script Language – A list of supported programming languages which allows the selection of language for scripts writing in the report (sets the Document.ScriptLanguage property).

**10** – Standard – A list of page formats which allows the selection of the page format for the report (sets the Page.PaperKind property).

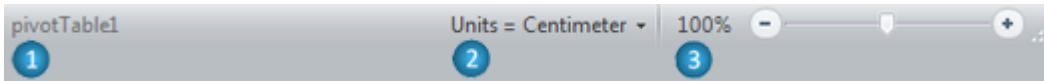
**11** – Custom – This field sets the page size for the report (sets the Page.CustomSize property).



- 12** – Orientation – Radio buttons set the page orientation for the report (sets the Page.Orientation property).
- 13** – Position – A list of values of headlines positions which allows the selection of the position in the template for headlines display with page numbers (top, bottom headlines).
- 14** – Align – A list of alignment values allows the selection of number text alignment for display in headlines.
- 15** – OK – This button shut downs Wizard work, creates a template with selected parameters, and opens it in Report Designer.
- 16** – Cancel – This buttons completes Wizard work.
- 20** – Pivot Table Settings – Pivot table settings tab.
- 21** – Data source – This button opens the tree for data source selection for pivot table, the field displays the selected data source.
- 23** – Add – This button adds a new field to the Nested fields list.
- 24** – Remove – This button removes the selected field from the Nested fields list.
- 25** – Configure layout – This button opens Source Fields editor for editing of pivot table data.
- 26** – Table Caption – This field sets the table title (sets PivotTable.TableCaption property).
- 28** – Show Single Fact – A flag which sets the display of fact column title (sets the PivotTable.ShowSingleFact property).
- 29** – Show Repeat Text – A flag which sets the repetition of the text on a new page (sets the PivotTable.ShowRepeatText property).
- 30** – Dimension Every Page – A flag which sets the headline output on each page (sets the PivotTable.DimensionOnEveryPage property).
- 31** – Auto Column Width – A flag which sets the default column width (sets the PivotTable.AutoColumnWidth property).
- 32** – Row Height – A counter which sets the row height (sets the PivotTable.RowHeight property).
- 33** – Auto Row Height Mode – A list which sets the auto height mode of the row (sets the PivotTable.AutoRowHeight property).

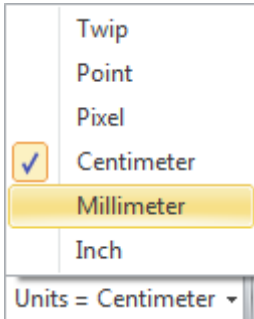


## Status Bar

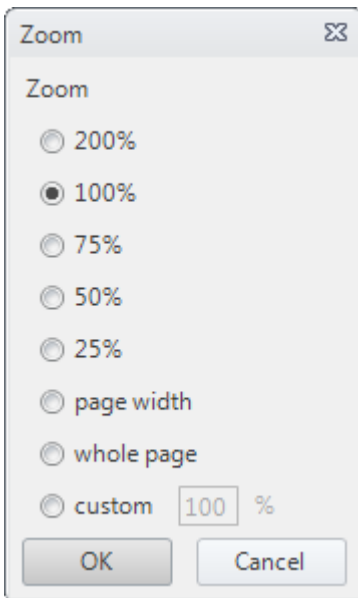


**1** – The Name of object, which is currently located under cursor.

**2** – The control for setting Metric Unit.



**5** – The button opens “Zoom” dialog to manually input Zoom value.



The Zoom slider sets zoom values.



## Hotkeys

Hotkey	Description
<b>Designer</b>	
F2	Switch the Select Mode on/off
F3	Switch the Pan Mode on/off
F4	Switch the Zoom In Mode on/off
F5	Switch the Zoom Out Mode on/off
F6	Switch the Zoom in Rectangle Mode on/off
F7	Switch the Dynamic Zoom Mode on/off
F10	Switch the Menu Mode on/off
Space + Drag Mouse	Switch the Pan Mode on/off
Ctrl + Mouse Wheel	Zoom
<b>Clipboard</b>	
Ctrl + Insert	Copy the selection and put it on the clipboard
Ctrl + C	
Shift + Insert	Paste the contents of the clipboard
Ctrl + V	
Shift + Delete	Cut the selection and put it on the clipboard
Ctrl + X	
Ctrl + Drag Mouse	Copy the selection
Ctrl + Alt + V	Paste the contents of the clipboard to the new page
<b>Template</b>	
Ctrl + N	Create a new template
Ctrl + O	Load a template or document from file
Ctrl + S	Save changes in a template
<b>Pages</b>	
Ctrl + Page Up	Show next page
Ctrl + Page Down	Show previous page
<b>Selecting</b>	
Ctrl + A	Select all
Ctrl + Shift + A	<ol style="list-style-type: none"> <li>1. Select all elements in the current container (and child elements)</li> <li>2. press A – add parent container</li> <li>3. press A – add all brothers of the parent (and their child elements)</li> <li>4. press A – add parent of the parent</li> <li>5. ...</li> </ol>
Drag Move	Select all elements that are at least 50% covered by the selection rectangle
Shift + Mouse Click	Add to selection/ remove selection
Ctrl + Mouse Click	
<b>Objects management</b>	
Ctrl + Enter	Call an Editor for selected element (applicable to TextBox, AdvancedText, RichText, Picture, Chart, Widget)
Enter	
Ctrl + Enter	Close an Editor, applying changes
Repeated Click	If many elements are overlapped, then when you click the overlapping



	area repeatedly the element with lower z-index will be selected (after the last element is selected the most top element will be selected again)
<b>Objects properties</b>	
Ctrl + B	Set Font.Bold property (applicable to TextBox, AdvancedText, BarCode)
Ctrl + I	Set Font.Italic property (applicable to TextBox, AdvancedText, BarCode)
Ctrl + U	Set Font.Underline property (applicable to TextBox, AdvancedText, BarCode)
Ctrl + Shift + >	Increase font size (applicable to TextBox, AdvancedText, BarCode)
Ctrl + Shift + <	Decrease font size (applicable to TextBox, AdvancedText, BarCode)
Ctrl + ]	Increase font size of a point (applicable to TextBox, AdvancedText, BarCode)
Ctrl + [	Decrease font size of a point (applicable to TextBox, AdvancedText, BarCode)
Ctrl + L	Align text to the left (applicable to TextBox, AdvancedText)
Ctrl + R	Align text to the right (applicable to TextBox, AdvancedText)
Ctrl + E	Center text (applicable to TextBox, AdvancedText)
Ctrl + Shift + F	Call Font dialog for setting Font property (applicable to TextBox, AdvancedText, BarCode)
Ctrl + Shift + B	Call Border Editor for setting Border property (applicable to objects, which have Border property)
Ctrl + Shift + M	Call Margins Editor for setting Margins property (applicable to objects, which have Margins property)
Ctrl + Shift + L	Call Fill Editor for setting Fill property (applicable to objects, which have Fill property)
Ctrl + Shift + T	Call Format Editor for setting Format property (applicable to TextBox, AdvancedText)
Ctrl + Shift + ]	Set Angle property, clockwise rotation (applicable to objects, which have Angle property)
Ctrl + Shift + [	Set Angle property, counterclockwise rotation (applicable to objects, which have Angle property)
Ctrl + Alt + F	Bring to front
Ctrl + Alt + B	Send to back





## Template Objects

### Classification

All the report elements can be divided into two types:

Bands (DataBand, PageHeader, PageFooter, Header, Footer, GroupBand, PageOverlay, CrossBand, BandContainer, SideBySide, SubReport, Content, PivotTable)

Visual elements (TextBox, AdvancedText, Picture, Shape, ZipCode, BarCode, RichText, WinFormsControl)

### Bands

The bands are containers for other elements; they don't have a graphical presentation in the report and serve for defining output methods of included elements. Bands are often used as containers for other bands. In this case, output methods of a contained element depend on all the container bands of the element. Band width is a permanent value and always equals page width. The location of the elements on the page is defined during report generation and depends on band order in the report, bands content and the properties of the Render group.

(More information about band properties)

### DataBand

The band for a multiple output of embedded components. Every line corresponds to one record in the data source. Data source for DataBand is specified by the DataSource property. If the data source is not specified, a number of repetitions are set in InstanceCount property.

The following bands can be used as container for DataBand: Page, DataBand, BandContainer, GroupBand and SideBySide.

(More information about using DataBand for building lists)

### Detail

The band provides access to the data from the data source of a container band and for output visual elements in various bands. It specifies the height that should be left for the output of the elements embedded into this band.

The following bands can be used as container for Detail: Page, DataBand, BandContainer, GroupBand and SideBySide.

(More information about Detail)

### PageHeader and PageFooter

The bands are used for creating top and bottom headlines of a page. These elements are containers and the spaces before and after the embedded elements in the final report will be specified by the space between these elements and top\bottom border of a band in a template. The location of PageHeader and PageFooter does not influence the location of the elements in the final report.

Only Page can be a container for PageHeader and PageFooter.

(More information about headlines usage for page design)

### Header and Footer

The bands allow creating a headline and a lower half of a page for the lists output by the DataBand and GroupBands. The contents of a Header will be displayed in the beginning of a list output by a GroupBand or in a headline of every group. The contents of Footer will be displayed at the end of a list or a group.

DataBand and GroupBand can be used as containers for Header and Footers.



(More information about the output of header and footer for the lists and grouped data)

### ***GroupBand***

The band is used for grouping data depending on a specified expression that is calculated during the report generation.

DataBand and GroupBand can be used as containers for GroupBand.

(More information about lists grouping)

### ***PageOverlay***

The band allows creating a bottom layer of a page. The space between the embedded elements and the beginning of a page in the final report is set by the indent of the top border of the band in the template. PageOverlay does not influence the location of other elements that allow locating elements over each other.

Only Page can be used as a container for PageOverlay.

(More information about page design using page overlay)

### ***CrossBand***

The band allows output of the embedded elements as a line LTR. The number of repetitions is specified either by the InstanceCount property or by a number of records in a DataSource. CrossBand band can be used for tables building.

The following bands can be used as containers for CrossBand: Header, Footer and Detail.

(More information about building horizontal lists and tables)

### ***BandContainer***

The band outputs its content once. Used for merging several bands in one connected SideBySide for the various variants of band merge.

The following bands can be used as containers for BandContainer: Page, DataBand, BandContainer, GroupBand and SideBySide.

### ***SideBySide***

The band is used for horizontal SideBySide output of the elements from different bands.

The following bands can be used as containers for SideBySide: Page, DataBand, BandContainer, GroupBand and SideBySide.

(More information about SideBySide report)

### ***SubReport***

The band allows adding embedded reports that will be generated together. Only bands (DataBand, Detail, etc.) will be added from template used as subreport.

The following bands can be used as containers for SubReport: Page, DataBand, BandContainer, GroupBand, and SideBySide.

(More information about building subreports)

### ***Content***

The band used in parental reports. This element specifies the place to locate the content of filial report.

The following bands can be used as containers for Content: Page, DataBand, BandContainer, GroupBand and SideBySide.

(More information about building master-reports)



### ***PivotTable***

The PivotTable band is used for creating cross reports based on static data and also for building aggregate tables. The result of using PivotTable element is a block in a report that contains visual presentation of a described cross table.

There is a special visual editor of a cross table that allows specifying data used for calculation. Scripts can be used for creating rules of calculation. This gives great opportunities for table settings starting from summing up of several fields from the data source and ending with various variants of grouping in the perspectives of required report.

The following bands can be used as containers for PivotTable: Page, DataBand, BandContainer, GroupBand and SideBySide.

(More information about building pivot tables)

### ***Visual Elements***

The elements that have graphical presentation in a report. Its location is specified by the containing bands. Size and displaying of the elements are specified by the properties.

(More information about element properties)

### ***TextBox***

The element outputs textual information. The information can either be static or be generated during the generation of a report.

(More information about text output)

### ***AdvancedText***

The element allows output of the information as a formatted text. There are options setting paragraph and text styles and using expressions directly in the marked text. Formatted text can be set using HTML-alike markup language and the subset of RTF format. Formatted Text Editor is used for this purpose. Static RTF text can be loaded. AdvancedText supports only some subset of RTF.

(More information about formatted text output)

### ***Picture***

The element allows output of an image. The images can be loaded from a hard driver or a data source.

(More information about image output)

### ***Shape***

The element allows output of various geometric figures.

More information about figures output)

### ***ZipCode***

The element allows output of a zip-code.

(More information about zip-codes output)

### ***BarCode***

The element allows output of barcodes of different types. There are several settings depending on a type of a used barcode.

(More information about barcodes output)

### ***RichText***

The element allows output of a text in RTF format. The editor allows choosing an RTF file.



(More information about RTF-text output)

### *WinFormsControl*

The element allows including Windows Forms control element into the report. Only the image of a control element will be included into the report; it is not available for using its functions.



## User Guide

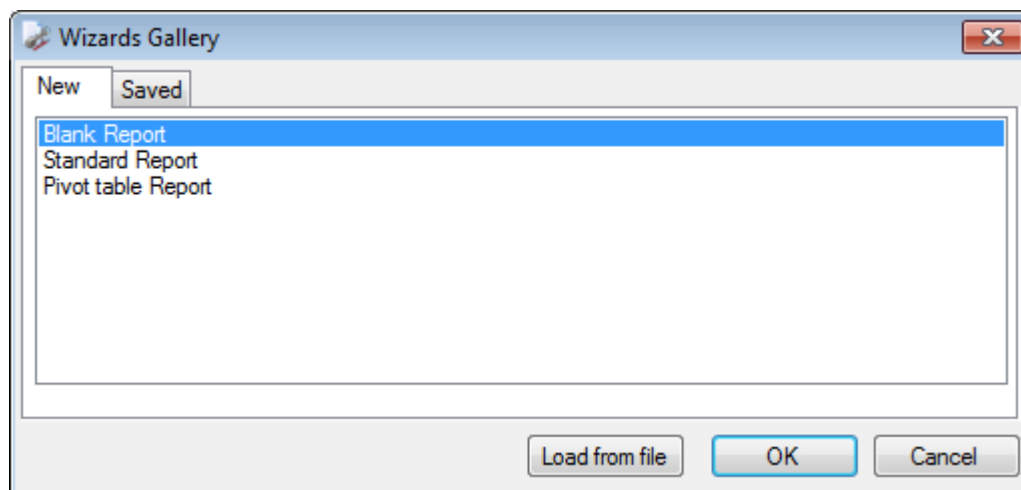
### Template

Report template is a number of pages including bands and objects.

#### Template Controls

##### Template Creation

Run "File\New" command of the main menu or press "New document" button on the Toolbox to create a new template. That will open Wizards Gallery where user can choose the way of template creating.



Items	Descriptions
Blank Report	Creates blank template
Standard Report	Opens Standard Wizard in order to create a standard template (a list or data cards)
Pivot table Report	Opens Pivotal Table Wizard in order to create a report with a Pivot Table

##### Template Opening

Run "File\Open" command of the main menu or press "Open Template" button on the Toolbox to open a dialog window. In this window one can choose a report to open. One can also double-click .rst template file that is associated with Report Design application.

##### Template Saving

Run "File\Save" command of the main menu or press "Save Template" button on the Toolbox to save a template. If the document has already been saved, the same file would be modified. If a new template is edited, Save Template dialog window appears to choose a folder and file name of a template. "File\Save as" command always calls Save Template dialog.

##### Template Generation

Run "File\Preview" Report command of the main menu or press "Preview Report" button on the Toolbox to generate a template.

(More information about Template Generation)

#### Template Properties

Template properties can be set with the help of Property Grid. Choose Document in the Object List or click the "Document Properties" on Quick Access Toolbar to display template properties in Property Grid.



## Graphical Settings

*GraphicsSettings* group includes the following properties:

Properties	Descriptions
CompositingQuality	set quality level, using in the compositing process
InterpolationMode	set interpolation mode
SmoothingMode	set smoothing mode
TextRenderingHint	set text visualization mode

## Elements Displaying and Grid

*ShowObjectBounds* property allows setting display elements borders in the template.

Property	Report Template								
<table border="1"> <tr><td colspan="2"><b>Appearance</b></td></tr> <tr><td>GraphicsSettings</td><td>PerpetuumSoft.Reporting.DOM.G</td></tr> <tr><td>MasterReport</td><td></td></tr> <tr><td>ShowObjectBounds</td><td>True</td></tr> </table>	<b>Appearance</b>		GraphicsSettings	PerpetuumSoft.Reporting.DOM.G	MasterReport		ShowObjectBounds	True	
<b>Appearance</b>									
GraphicsSettings	PerpetuumSoft.Reporting.DOM.G								
MasterReport									
ShowObjectBounds	True								
<table border="1"> <tr><td colspan="2"><b>Appearance</b></td></tr> <tr><td>GraphicsSettings</td><td>PerpetuumSoft.Reporting.DOM.G</td></tr> <tr><td>MasterReport</td><td></td></tr> <tr><td>ShowObjectBounds</td><td>False</td></tr> </table>	<b>Appearance</b>		GraphicsSettings	PerpetuumSoft.Reporting.DOM.G	MasterReport		ShowObjectBounds	False	
<b>Appearance</b>									
GraphicsSettings	PerpetuumSoft.Reporting.DOM.G								
MasterReport									
ShowObjectBounds	False								

*ShowGrid* property defines if the Grid is displayed or not, *GridStep* property sets grid wideness.

Property	Report Template								
<table border="1"> <tr><td colspan="2"><b>Grid Settings</b></td></tr> <tr><td>GridStep</td><td>0,25 cm</td></tr> <tr><td>ShowGrid</td><td>False</td></tr> <tr><td>SnapToGrid</td><td>True</td></tr> </table>	<b>Grid Settings</b>		GridStep	0,25 cm	ShowGrid	False	SnapToGrid	True	
<b>Grid Settings</b>									
GridStep	0,25 cm								
ShowGrid	False								
SnapToGrid	True								
<table border="1"> <tr><td colspan="2"><b>Grid Settings</b></td></tr> <tr><td>GridStep</td><td>0,25 cm</td></tr> <tr><td>ShowGrid</td><td>True</td></tr> <tr><td>SnapToGrid</td><td>True</td></tr> </table>	<b>Grid Settings</b>		GridStep	0,25 cm	ShowGrid	True	SnapToGrid	True	
<b>Grid Settings</b>									
GridStep	0,25 cm								
ShowGrid	True								
SnapToGrid	True								
<table border="1"> <tr><td colspan="2"><b>Grid Settings</b></td></tr> <tr><td>GridStep</td><td>0,5 cm</td></tr> <tr><td>ShowGrid</td><td>True</td></tr> <tr><td>SnapToGrid</td><td>True</td></tr> </table>	<b>Grid Settings</b>		GridStep	0,5 cm	ShowGrid	True	SnapToGrid	True	
<b>Grid Settings</b>									
GridStep	0,5 cm								
ShowGrid	True								
SnapToGrid	True								

*SnapToGrid* property is used for "sticking" elements to the grid when ones are created or when their sizes and location are changed.

One can use the corresponding buttons on the ToolBar for editing of these properties.

## Measurement Units

*MetricUnit* property is used for setting measurement units for the whole template.

<b>Misc</b>	
DataSources	(Collection)
Description	
MetricUnit	Centimeter
Parameters	Inch
Title	Millimeter
<b>Render</b>	
StyleSheet	Pixel
<b>Script</b>	
CommonScript	Twip
CompilerVersion	



### Template Data Sources

Application can be a data source, but there is an opportunity to set the data sources for the current template in Report Designer using the collection of built-in data sources – `Document.DataSources` property. In order to set this property `Document Data Sources` editor is used.

(More information about Data Sources)

### Template Parameters

Report parameters can be set with the help of *Parameters* collection.

(More information about Parameters)



## Page

Report template is a set of pages. Every page contains elements displaying information. A default empty report contains one page.

### Page Properties

Page Properties can be set with the help of Property Grid. Select Page in the Object List to display template properties in Property Grid.

#### Page Size

*PaperKind* property is used to set page size; it allows you to choose one of the standard page formats, e.g. A4 (default size), B5, Letter, etc. Page layout can be set for those page sizes – vertical or horizontal. In this case Size and *TemplateSize* properties gain chosen format settings and the properties cannot be edited.

Page Settings	
CustomSize	0; 0 cm
Margins	<b>1; 1,2; 1,5; 1 cm</b>
Orientation	Portrait
PaperKind	<b>B4</b>
TemplateSize	25; 35,3 cm

Select "Custom" value of the *PaperKind* property and set page size parameters of the Custom Size property if you want to create a non-standard paper size.

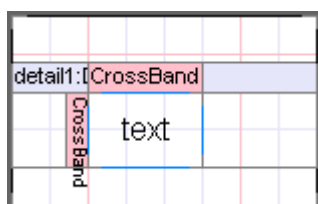
Page Settings	
CustomSize	<b>5; 5 cm</b>
X	<b>5 cm</b>
Y	<b>5 cm</b>
Margins	<b>1; 1,2; 1,5; 1 cm</b>
Orientation	Portrait
PaperKind	<b>Custom</b>
TemplateSize	5; 5 cm

#### Page Margins

Page Margins set the indent between page limits and the elements placed in the bands – containers. *Margins* property sets page margins. Margins are marked with the red line in the template.

Sample demonstrating use of Page Margins:

Report template:

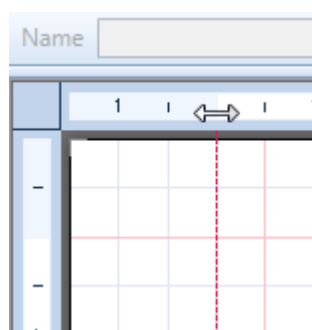


crossBand1.InstanceCount = 2



Property	Report																
<table border="1"> <tr><td colspan="2"><b>Page Settings</b></td></tr> <tr><td>CustomSize</td><td>0; 0 cm</td></tr> <tr><td><b>Margins</b></td><td><b>0.5; 0.5; 1; 1 cm</b></td></tr> <tr><td>Bottom</td><td><b>0.5 cm</b></td></tr> <tr><td>Left</td><td><b>1 cm</b></td></tr> <tr><td>Right</td><td><b>1 cm</b></td></tr> <tr><td>Top</td><td><b>0.5 cm</b></td></tr> <tr><td>Orientation</td><td>Portrait</td></tr> </table>	<b>Page Settings</b>		CustomSize	0; 0 cm	<b>Margins</b>	<b>0.5; 0.5; 1; 1 cm</b>	Bottom	<b>0.5 cm</b>	Left	<b>1 cm</b>	Right	<b>1 cm</b>	Top	<b>0.5 cm</b>	Orientation	Portrait	
<b>Page Settings</b>																	
CustomSize	0; 0 cm																
<b>Margins</b>	<b>0.5; 0.5; 1; 1 cm</b>																
Bottom	<b>0.5 cm</b>																
Left	<b>1 cm</b>																
Right	<b>1 cm</b>																
Top	<b>0.5 cm</b>																
Orientation	Portrait																
<table border="1"> <tr><td colspan="2"><b>Page Settings</b></td></tr> <tr><td>CustomSize</td><td>0; 0 cm</td></tr> <tr><td><b>Margins</b></td><td><b>0; 0; 1; 1 cm</b></td></tr> <tr><td>Bottom</td><td><b>0 cm</b></td></tr> <tr><td>Left</td><td><b>1 cm</b></td></tr> <tr><td>Right</td><td><b>1 cm</b></td></tr> <tr><td>Top</td><td><b>0 cm</b></td></tr> <tr><td>Orientation</td><td>Portrait</td></tr> </table>	<b>Page Settings</b>		CustomSize	0; 0 cm	<b>Margins</b>	<b>0; 0; 1; 1 cm</b>	Bottom	<b>0 cm</b>	Left	<b>1 cm</b>	Right	<b>1 cm</b>	Top	<b>0 cm</b>	Orientation	Portrait	
<b>Page Settings</b>																	
CustomSize	0; 0 cm																
<b>Margins</b>	<b>0; 0; 1; 1 cm</b>																
Bottom	<b>0 cm</b>																
Left	<b>1 cm</b>																
Right	<b>1 cm</b>																
Top	<b>0 cm</b>																
Orientation	Portrait																
<table border="1"> <tr><td colspan="2"><b>Page Settings</b></td></tr> <tr><td>CustomSize</td><td>0; 0 cm</td></tr> <tr><td><b>Margins</b></td><td><b>0; 0; 0; 0 cm</b></td></tr> <tr><td>Bottom</td><td><b>0 cm</b></td></tr> <tr><td>Left</td><td><b>0 cm</b></td></tr> <tr><td>Right</td><td><b>0 cm</b></td></tr> <tr><td>Top</td><td><b>0 cm</b></td></tr> <tr><td>Orientation</td><td>Portrait</td></tr> </table>	<b>Page Settings</b>		CustomSize	0; 0 cm	<b>Margins</b>	<b>0; 0; 0; 0 cm</b>	Bottom	<b>0 cm</b>	Left	<b>0 cm</b>	Right	<b>0 cm</b>	Top	<b>0 cm</b>	Orientation	Portrait	
<b>Page Settings</b>																	
CustomSize	0; 0 cm																
<b>Margins</b>	<b>0; 0; 0; 0 cm</b>																
Bottom	<b>0 cm</b>																
Left	<b>0 cm</b>																
Right	<b>0 cm</b>																
Top	<b>0 cm</b>																
Orientation	Portrait																

It is possible to change the Page Margins using the Ruler.



### Page Style

*StyleName* property allows setting a united style for all of the elements on the page. Create a style in the Style Sheet Editor and choose your style as a *StyleName* property parameter. The chosen style will be set as default for all the elements and bands added to the page. Being edited, the style will be applied to all the page elements, if their own style is not changed separately.

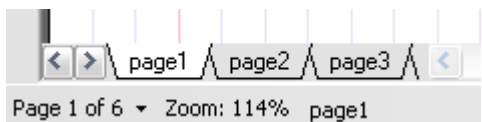
(More information about styles)



### Multipage Report

You can add as many pages to your report as it is required, e.g. in the report with a cover page. The ready report is a combination of the reports formed according to the page templates. Parts of the report will be displayed in the same order as the pages order in the template.

Click the "New Page" button on the "Insert" tab to add a page to the report. When the page is removed, all the objects contained on this page are removed too. One can switch between pages using the tabs in the bottom left corner of the Report Designer.

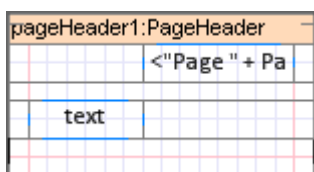


### Page Headlines

PageHeader and PageFooter are used to set upper and the bottom headlines respectively. Band contents will be displayed in the upper and the bottom part of the page respectively, regardless of the band location on the page. Mode property sets the headliner generation mode – for every page, even\odd pages.

Sample demonstrating use of Page headlines:

Report Template:



textBoxPageHeader.Value = "Page " + PageNumber

Mode Property	Report										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>Mode</td><td>AllPages</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	Mode	AllPages	Visible	True	
<b>Render</b>											
CanGrow	False										
CanShrink	False										
Mode	AllPages										
Visible	True										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>Mode</td><td>OddPages</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	Mode	OddPages	Visible	True	
<b>Render</b>											
CanGrow	False										
CanShrink	False										
Mode	OddPages										
Visible	True										



Render	
CanGrow	False
CanShrink	False
Mode	EvenPages <input type="button" value="v"/>
Visible	True

text

---

Page 2

text



### Page Numeration

*PageNumber* property is used for getting the page number; *PageCount* property contains the total amount of the pages in a report. During the first pass, total page amount is unknown and *PageCount* property will be equal to *PageNumber* property. Set *DoublePass* property of the Document object to "True" for *PageCount* property to keep the total amount of the pages.

Example displaying page number in the header:

```

dataBand1.InstanceCount = 3;

textBoxHeader.Value = "Page " + PageNumber.ToString() + " of " + PageCount.ToString();

textBoxDetail.Value = "Line Number " + dataBand1.LineNumber().ToString();

```

Report Template	Report
pageHeader1:PageHeader <"Page " + PageNumber.ToString() + " of " + PageCount.ToString();	Page 1 of 2
dataBand1:DataBand	Line Number 1
detail1:Detail <"Line Number "	Line Number 2
end of dataBand1	
	Page 2 of 2
	Line Number 3

### Page Overlay

*PageOverlay* is used when there is a necessity to display data on the background. Element location will be calculated according to the band location regardless of *PageOverlay* location on the page. The band does not affect location of the other bands; other bands contents will be displayed over the *PageOverlay* elements.

Report Template	Report
pageOverlay1:PageOverlay text	text
pageOverlay1:PageOverlay text	text
pageOverlay1:PageOverlay text	text

Mode property sets the overlay generation mode – for every page, even\odd pages.

## Objects' Controls

### Object Addition

The "Insert" tab is used to add objects. Then set the mouse pointer on the page and left-click to add an object. If you click on some band, the program checks if this band can contain the object you try to add (More information about available band containers). If the band can contain this object, it is added to the template, inside the band, otherwise an error message is shown.

TextBox elements for data source outlet can be added to the template by drag and drop of the appropriate element from the Data Tree of the Tool Window from the Data Sources tab. TextBox.Value will be set automatically.

### Object Deletion

There are several ways to delete an object: left-click (or select it in the Object list or in the Document Tree in the Tool Window) and press "Delete" button on the keyboard (or use the contextual menu item). When you delete a band, all the contained objects will be deleted too.

### Object Properties

Object properties can be set with the help of Property Grid. Select the object in the template to see its properties displayed in the Property Grid. Property Grid shows common properties of the elements when several of them are selected.

(More information about objects' properties)

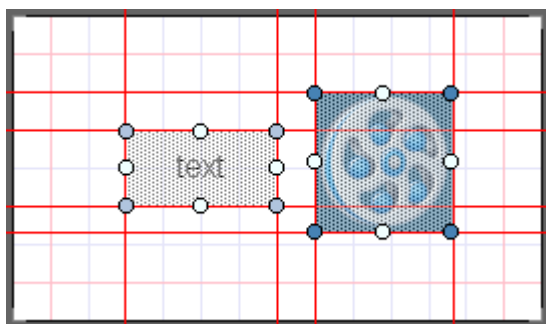
### Document Tree

Document Tree contains the list of all document objects and shows its structure. One can select and delete object, change hierarchy (dragging the object to the different band) and change object order in the Document Tree.

### Objects' Location

Objects' location in the template is set with the *Size* property. The higher object is displayed when two overlay each other. Use Align Editor to align several objects relatively to each other.

One can change the location of several chosen objects included into the same container simultaneously with the mouse pointer.



It is possible to change object position with the navigation buttons on the keyboard. Pressed Ctrl key prevents objects from "sticking" to the grid.


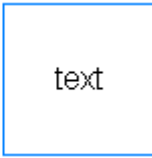



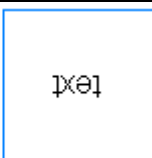
## Objects' Properties

Object properties define object appearance and behavior when generating reports. Object properties can be set by Property Grid. To display object properties in the Property Grid one needs to select an element in a template with left mouse button click or select it in the Object List. Properties default values have regular font, edited properties values are displayed in bold. One may select several objects by pressing the Shift button. This way there are general properties of the selected objects on Property Grid. All object properties may be divided into several groups:

Groups	Descriptions
Appearance	properties defining objects display and appearance
Data	properties defining data processing
Design	properties defining objects display when editing a template in Report Designer
Layout	properties defining objects location and size
Misc	properties for objects which are built in the report
Navigation	properties which allow reports navigation creation
PageSetting	page properties
Render	properties defining rules of objects behavior during reports generation
Script	scripts – objects processing

### Angle (Appearance)

The property is used to set a rotate angle of the contained element relative to its area. Acceptable values – 0, 90, 180 and 270. Examples of using the rotate angle for the Pictures and TextBox elements:

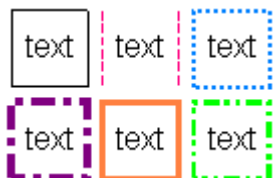
Property	Report														
<table border="1"> <tr> <td colspan="2"><input type="checkbox"/> <b>Appearance</b></td> </tr> <tr> <td>Angle</td> <td>0</td> </tr> <tr> <td>Border</td> <td><b>Border</b></td> </tr> <tr> <td>Fill</td> <td>[None]</td> </tr> <tr> <td><input type="checkbox"/> Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	<input type="checkbox"/> <b>Appearance</b>		Angle	0	Border	<b>Border</b>	Fill	[None]	<input type="checkbox"/> Margins	0; 0; 0; 0 cm	StyleName		ToolTip		 
<input type="checkbox"/> <b>Appearance</b>															
Angle	0														
Border	<b>Border</b>														
Fill	[None]														
<input type="checkbox"/> Margins	0; 0; 0; 0 cm														
StyleName															
ToolTip															
<table border="1"> <tr> <td colspan="2"><input type="checkbox"/> <b>Appearance</b></td> </tr> <tr> <td>Angle</td> <td><b>90</b></td> </tr> <tr> <td>Border</td> <td><b>Border</b></td> </tr> <tr> <td>Fill</td> <td>[None]</td> </tr> <tr> <td><input type="checkbox"/> Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	<input type="checkbox"/> <b>Appearance</b>		Angle	<b>90</b>	Border	<b>Border</b>	Fill	[None]	<input type="checkbox"/> Margins	0; 0; 0; 0 cm	StyleName		ToolTip		 
<input type="checkbox"/> <b>Appearance</b>															
Angle	<b>90</b>														
Border	<b>Border</b>														
Fill	[None]														
<input type="checkbox"/> Margins	0; 0; 0; 0 cm														
StyleName															
ToolTip															
<table border="1"> <tr> <td colspan="2"><input type="checkbox"/> <b>Appearance</b></td> </tr> <tr> <td>Angle</td> <td><b>180</b></td> </tr> <tr> <td>Border</td> <td><b>Border</b></td> </tr> <tr> <td>Fill</td> <td>[None]</td> </tr> <tr> <td><input type="checkbox"/> Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	<input type="checkbox"/> <b>Appearance</b>		Angle	<b>180</b>	Border	<b>Border</b>	Fill	[None]	<input type="checkbox"/> Margins	0; 0; 0; 0 cm	StyleName		ToolTip		 
<input type="checkbox"/> <b>Appearance</b>															
Angle	<b>180</b>														
Border	<b>Border</b>														
Fill	[None]														
<input type="checkbox"/> Margins	0; 0; 0; 0 cm														
StyleName															
ToolTip															



### Border (Appearance)

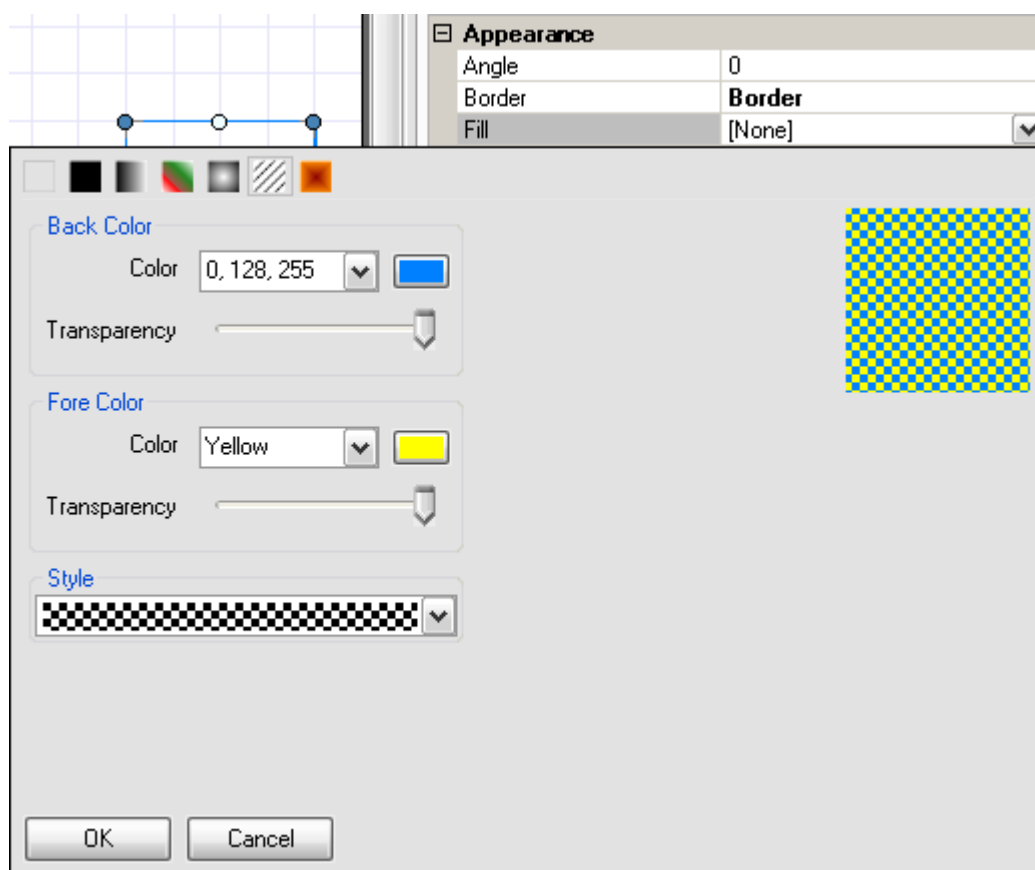
The property allows setting the display of element borders and lines draw style – type, thickness and color (Border Editor).

Border examples for the TextBox element:

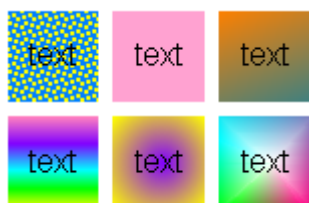


### Fill (Appearance)

The property allows setting color and style of object area fill. In order to edit the property one may use the fill editor where one may set fill type, color, transparency and additional parameters.



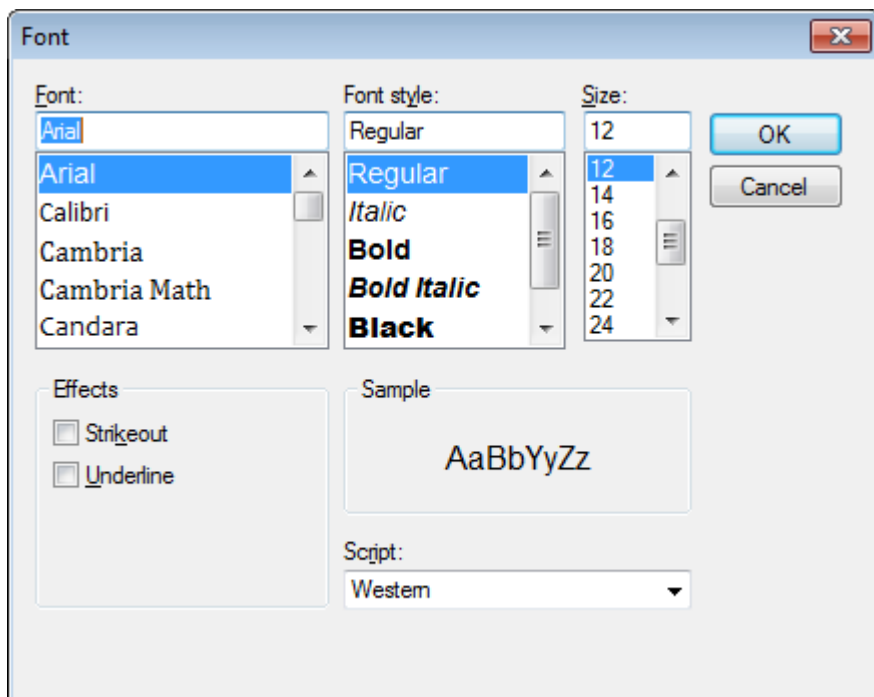
Examples of using fill for the TextBox element:



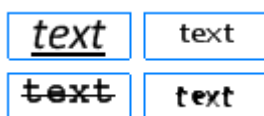


### Font (Appearance)

The property allows setting font type for elements. There is the font editor for properties editing. One may use it to set font, font style and size.



Sample demonstrating use of TextBox element fonts:



Note: If the text that is output by the AdvancedText and its font is set in the Formatted Text Editor, Font property is ignored.





### Margins (Appearance)

Property allows setting object margins.

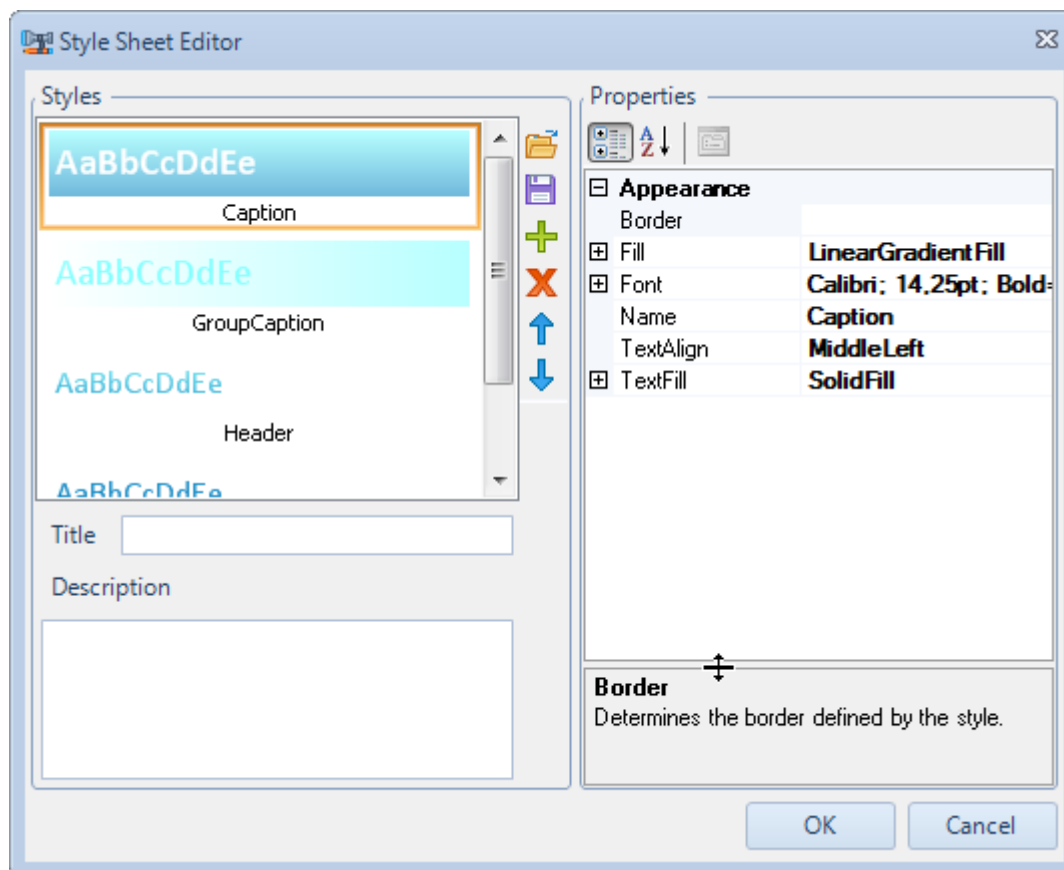
Example of changing margins of the Picture object:

Property	Report																						
<table border="1"> <tr><td colspan="2"><b>Appearance</b></td></tr> <tr><td>Angle</td><td>0</td></tr> <tr><td>Border</td><td></td></tr> <tr><td>Fill</td><td>[None]</td></tr> <tr><td>Image</td><td> System.Drawing.Bitmap</td></tr> <tr><td>ImageAlign</td><td>TopLeft</td></tr> <tr><td colspan="2"><b>Margins</b></td></tr> <tr><td>Bottom</td><td>0 cm</td></tr> <tr><td>Left</td><td>0 cm</td></tr> <tr><td>Right</td><td>0 cm</td></tr> <tr><td>Top</td><td>0 cm</td></tr> </table>	<b>Appearance</b>		Angle	0	Border		Fill	[None]	Image	System.Drawing.Bitmap	ImageAlign	TopLeft	<b>Margins</b>		Bottom	0 cm	Left	0 cm	Right	0 cm	Top	0 cm	
<b>Appearance</b>																							
Angle	0																						
Border																							
Fill	[None]																						
Image	System.Drawing.Bitmap																						
ImageAlign	TopLeft																						
<b>Margins</b>																							
Bottom	0 cm																						
Left	0 cm																						
Right	0 cm																						
Top	0 cm																						
<table border="1"> <tr><td colspan="2"><b>Appearance</b></td></tr> <tr><td>Angle</td><td>0</td></tr> <tr><td>Border</td><td></td></tr> <tr><td>Fill</td><td>[None]</td></tr> <tr><td>Image</td><td> System.Drawing.Bitmap</td></tr> <tr><td>ImageAlign</td><td>TopLeft</td></tr> <tr><td colspan="2"><b>Margins</b></td></tr> <tr><td>Bottom</td><td>0,2 cm</td></tr> <tr><td>Left</td><td>0,2 cm</td></tr> <tr><td>Right</td><td>0,2 cm</td></tr> <tr><td>Top</td><td>0,2 cm</td></tr> </table>	<b>Appearance</b>		Angle	0	Border		Fill	[None]	Image	System.Drawing.Bitmap	ImageAlign	TopLeft	<b>Margins</b>		Bottom	0,2 cm	Left	0,2 cm	Right	0,2 cm	Top	0,2 cm	
<b>Appearance</b>																							
Angle	0																						
Border																							
Fill	[None]																						
Image	System.Drawing.Bitmap																						
ImageAlign	TopLeft																						
<b>Margins</b>																							
Bottom	0,2 cm																						
Left	0,2 cm																						
Right	0,2 cm																						
Top	0,2 cm																						

### StyleName (Appearance)

Property quickly edits object style, applying existing style to one or several objects.

“Edit Style” button on the “Home” tab opens Style Sheet Editor.



Use “Open” and “Save” buttons to open and save styles. A file has XML format and .rss filename extension. “Add Style” and “Remove Style” buttons add and remove styles.

The following style properties are available:

Properties	Descriptions
Border	Element border
Font	Style of the element’s text
Name	Style name
TextAlign	Alignment of the element’s text
TextFill	Fill of the element’s text

Set required style name as a parameter of the StyleName property to apply this style to the element.

Example demonstrating use of StyleName property for TextBox and Shape elements.

Report Template		Created style															
		<table border="1"> <tr> <td colspan="2">[-] Appearance</td> </tr> <tr> <td>Border</td> <td><b>Border</b></td> </tr> <tr> <td>+</td> <td>Fill <b>LinearGradientFill</b></td> </tr> <tr> <td>+</td> <td>Font <b>Calibri; 20,25pt; Bol</b></td> </tr> <tr> <td>Name</td> <td><b>New Style</b></td> </tr> <tr> <td>TextAlign</td> <td>MiddleCenter</td> </tr> <tr> <td>+</td> <td>TextFill <b>LinearGradientFill</b></td> </tr> </table>		[-] Appearance		Border	<b>Border</b>	+	Fill <b>LinearGradientFill</b>	+	Font <b>Calibri; 20,25pt; Bol</b>	Name	<b>New Style</b>	TextAlign	MiddleCenter	+	TextFill <b>LinearGradientFill</b>
[-] Appearance																	
Border	<b>Border</b>																
+	Fill <b>LinearGradientFill</b>																
+	Font <b>Calibri; 20,25pt; Bol</b>																
Name	<b>New Style</b>																
TextAlign	MiddleCenter																
+	TextFill <b>LinearGradientFill</b>																
Property																	
<table border="1"> <tr> <td colspan="2">[-] Appearance</td> </tr> <tr> <td>Fill</td> <td>[None]</td> </tr> <tr> <td>StyleName</td> <td>▼</td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>		[-] Appearance		Fill	[None]	StyleName	▼	ToolTip									
[-] Appearance																	
Fill	[None]																
StyleName	▼																
ToolTip																	
<table border="1"> <tr> <td colspan="2">[-] Appearance</td> </tr> <tr> <td>+</td> <td>Fill LinearGradientFill</td> </tr> <tr> <td>StyleName</td> <td><b>New Style</b></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>		[-] Appearance		+	Fill LinearGradientFill	StyleName	<b>New Style</b>	ToolTip									
[-] Appearance																	
+	Fill LinearGradientFill																
StyleName	<b>New Style</b>																
ToolTip																	

Note 1: When a style is applied to a container, it applies to all the contained objects. Appropriate properties of the objects receive the same values the style has (only if they have default value, i.e. not changed manually). The values different from the default are marked with the bold font.

Note 2: If the text that is output by the AdvancedText and its font is set in the Formatted text editor, Font property is ignored.

### TextAlign (Appearance)

Property aligns text relatively to the object space.

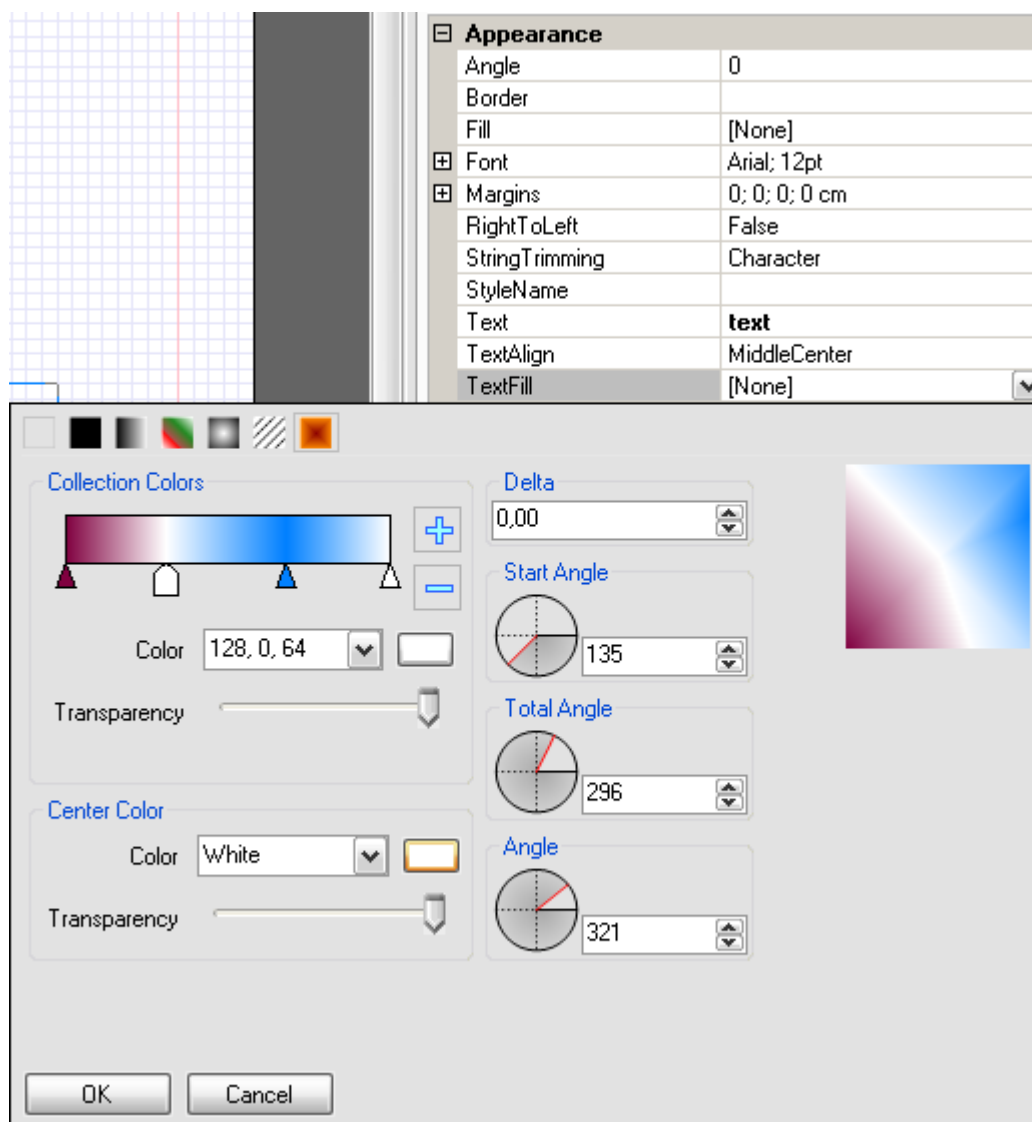
TextBox element alignment example:

Property	Report												
<table border="1"> <tr><td>StyleName</td><td></td></tr> <tr><td>Text</td><td><b>text</b></td></tr> <tr><td>TextAlign</td><td><b>TopLeft</b></td></tr> <tr><td>TextFill</td><td>[None]</td></tr> <tr><td>TextFormat</td><td></td></tr> <tr><td>ToolTip</td><td></td></tr> </table>	StyleName		Text	<b>text</b>	TextAlign	<b>TopLeft</b>	TextFill	[None]	TextFormat		ToolTip		
StyleName													
Text	<b>text</b>												
TextAlign	<b>TopLeft</b>												
TextFill	[None]												
TextFormat													
ToolTip													
<table border="1"> <tr><td>StyleName</td><td></td></tr> <tr><td>Text</td><td><b>text</b></td></tr> <tr><td>TextAlign</td><td><b>MiddleCenter</b></td></tr> <tr><td>TextFill</td><td>[None]</td></tr> <tr><td>TextFormat</td><td></td></tr> <tr><td>ToolTip</td><td></td></tr> </table>	StyleName		Text	<b>text</b>	TextAlign	<b>MiddleCenter</b>	TextFill	[None]	TextFormat		ToolTip		
StyleName													
Text	<b>text</b>												
TextAlign	<b>MiddleCenter</b>												
TextFill	[None]												
TextFormat													
ToolTip													
<table border="1"> <tr><td>StyleName</td><td></td></tr> <tr><td>Text</td><td><b>text</b></td></tr> <tr><td>TextAlign</td><td><b>BottomRight</b></td></tr> <tr><td>TextFill</td><td>[None]</td></tr> <tr><td>TextFormat</td><td></td></tr> <tr><td>ToolTip</td><td></td></tr> </table>	StyleName		Text	<b>text</b>	TextAlign	<b>BottomRight</b>	TextFill	[None]	TextFormat		ToolTip		
StyleName													
Text	<b>text</b>												
TextAlign	<b>BottomRight</b>												
TextFill	[None]												
TextFormat													
ToolTip													

### TextFill (Appearance)

Property sets the text fill.

Fill editor helps to set the properties of the fill. Fill type, colors, transparency and additional parameters are available.



Examples demonstrating text fill for the TextBox element:

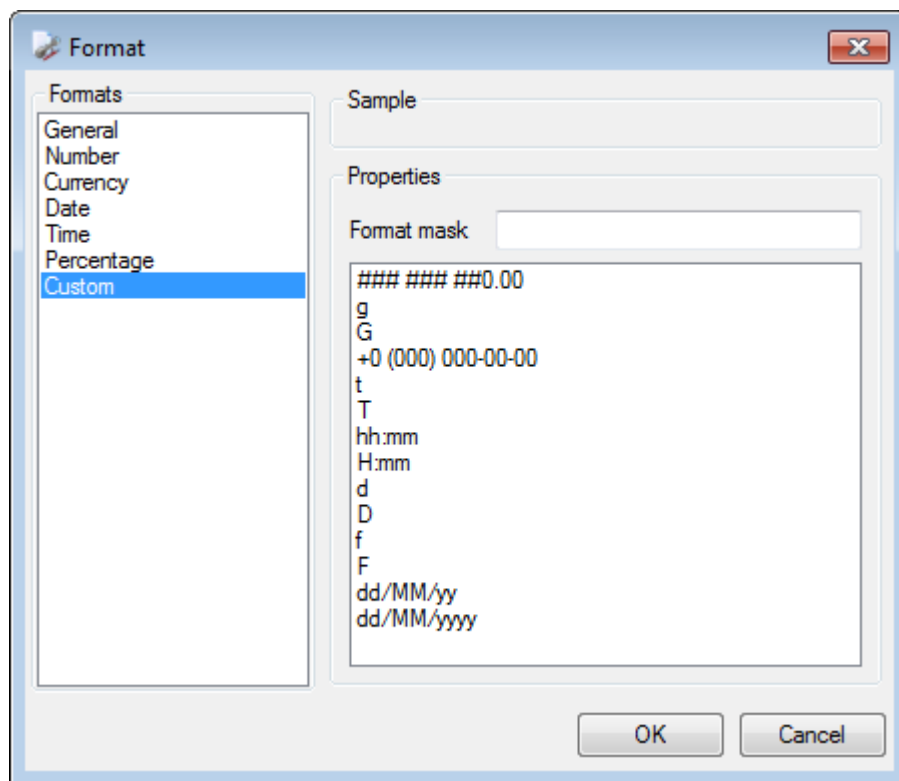


Note: If the text that is output by the AdvancedText and its font is set in the Formatted Text Editor, Font property is ignored.

### TextFormat (Appearance)

Property changes text in the set format: number, currency, date format, time format and percentage. Custom format can be created.

Setting text format in Format editor.



Example demonstrating text format applied to the TextBox element:

Property		Report
StyleName		
Text	<b>34.5</b>	£34.50
TextAlign	MiddleCenter	
TextFill	[None]	
TextFormat	<b>Currency</b>	
ToolTip		
StyleName		
Text	<b>02.17.2010</b>	17 February 2010
TextAlign	MiddleCenter	
TextFill	[None]	
TextFormat	<b>Date</b>	
ToolTip		
StyleName		
Text	<b>123456</b>	123,456.00
TextAlign	MiddleCenter	
TextFill	[None]	
TextFormat	<b>Number</b>	
ToolTip		

### ToolTip (Appearance)

Property creates a hint that appears when the element is pointed at.

Examples demonstrating use of ToolTip for the Shape element.

Property	Report														
<table border="1"> <tr><td colspan="2">[-] Appearance</td></tr> <tr><td>Fill</td><td>ConicalFill</td></tr> <tr><td>Line</td><td>0 Solid Black</td></tr> <tr><td>Shadow</td><td>Shadow</td></tr> <tr><td>ShapeStyle</td><td>StarShape</td></tr> <tr><td>StyleName</td><td></td></tr> <tr><td>ToolTip</td><td>StarShape</td></tr> </table>	[-] Appearance		Fill	ConicalFill	Line	0 Solid Black	Shadow	Shadow	ShapeStyle	StarShape	StyleName		ToolTip	StarShape	
[-] Appearance															
Fill	ConicalFill														
Line	0 Solid Black														
Shadow	Shadow														
ShapeStyle	StarShape														
StyleName															
ToolTip	StarShape														

### Tag (Data)

Property allows additional information of the object made by a user.

Tag usage example:

[-] Data	
Tag	additional information

### Name (Design)

Property sets the name of the object. Every object must have a unique name.

Setting name for BandContainer:

Properties	
newName BandContainer	
[-] Appearance	
StyleName	Normal
ToolTip	
[-] Data	
Tag	
[-] Design	
Name	newName
ShowCaption	True

### ShowBottom (Design)

Property allows setting the bottom line of the band visible in Designer mode.

Example demonstrating use of ShowBottom property for DataBand property:

Property	Report								
<table border="1"> <tr><td colspan="2">[-] Design</td></tr> <tr><td>Name</td><td>dataBand2</td></tr> <tr><td>ShowBottom</td><td>True</td></tr> <tr><td>ShowCaption</td><td>True</td></tr> </table>	[-] Design		Name	dataBand2	ShowBottom	True	ShowCaption	True	
[-] Design									
Name	dataBand2								
ShowBottom	True								
ShowCaption	True								
<table border="1"> <tr><td colspan="2">[-] Design</td></tr> <tr><td>Name</td><td>dataBand2</td></tr> <tr><td>ShowBottom</td><td>False</td></tr> <tr><td>ShowCaption</td><td>True</td></tr> </table>	[-] Design		Name	dataBand2	ShowBottom	False	ShowCaption	True	
[-] Design									
Name	dataBand2								
ShowBottom	False								
ShowCaption	True								

### ShowCaption (Design)

Property sets band header visibility in Designer mode.

Example demonstrating use of ShowCaption property for DataBand property:

Property	Report								
<table border="1"> <tr><td colspan="2"><b>Design</b></td></tr> <tr><td>Name</td><td>dataBand2</td></tr> <tr><td>ShowBottom</td><td>True</td></tr> <tr><td>ShowCaption</td><td>True</td></tr> </table>	<b>Design</b>		Name	dataBand2	ShowBottom	True	ShowCaption	True	
<b>Design</b>									
Name	dataBand2								
ShowBottom	True								
ShowCaption	True								
<table border="1"> <tr><td colspan="2"><b>Design</b></td></tr> <tr><td>Name</td><td>dataBand2</td></tr> <tr><td>ShowBottom</td><td>False</td></tr> <tr><td>ShowCaption</td><td>True</td></tr> </table>	<b>Design</b>		Name	dataBand2	ShowBottom	False	ShowCaption	True	
<b>Design</b>									
Name	dataBand2								
ShowBottom	False								
ShowCaption	True								

### Location (Layout)

Property sets upper left corner of the object relatively to the container.

Element position can be changed with the mouse pointer or on the Property grid.

Example of the changing element position for Shape element:

Property	Report												
<table border="1"> <tr><td colspan="2"><b>Layout</b></td></tr> <tr><td colspan="2"><b>Location</b></td></tr> <tr><td>X</td><td>0 cm</td></tr> <tr><td>Y</td><td>0 cm</td></tr> <tr><td colspan="2"><b>Size</b></td></tr> <tr><td></td><td>2; 2 cm</td></tr> </table>	<b>Layout</b>		<b>Location</b>		X	0 cm	Y	0 cm	<b>Size</b>			2; 2 cm	
<b>Layout</b>													
<b>Location</b>													
X	0 cm												
Y	0 cm												
<b>Size</b>													
	2; 2 cm												
<table border="1"> <tr><td colspan="2"><b>Layout</b></td></tr> <tr><td colspan="2"><b>Location</b></td></tr> <tr><td>X</td><td>1 cm</td></tr> <tr><td>Y</td><td>1 cm</td></tr> <tr><td colspan="2"><b>Size</b></td></tr> <tr><td></td><td>2; 2 cm</td></tr> </table>	<b>Layout</b>		<b>Location</b>		X	1 cm	Y	1 cm	<b>Size</b>			2; 2 cm	
<b>Layout</b>													
<b>Location</b>													
X	1 cm												
Y	1 cm												
<b>Size</b>													
	2; 2 cm												

Note 1: X coordinate equals 0 and cannot be changed.

Note 2: "Snap to Grid" option defines if the object "sticks" to the grid or not when its position is changed with the mouse pointer. Click the button on the Toolbar to turn on/off this option.



### Size (Layout)

Property sets object size.

Property changes when object borders are moved with the mouse pointer or from the Property grid.

Example of changing the element size for the Shape element:

Property	Report										
<table border="1"> <tr><td><input type="checkbox"/> <b>Layout</b></td><td></td></tr> <tr><td><input checked="" type="checkbox"/> Location</td><td>1; 1 cm</td></tr> <tr><td><input type="checkbox"/> Size</td><td>2; 2 cm</td></tr> <tr><td>    X</td><td>2 cm</td></tr> <tr><td>    Y</td><td>2 cm</td></tr> </table>	<input type="checkbox"/> <b>Layout</b>		<input checked="" type="checkbox"/> Location	1; 1 cm	<input type="checkbox"/> Size	2; 2 cm	X	2 cm	Y	2 cm	
<input type="checkbox"/> <b>Layout</b>											
<input checked="" type="checkbox"/> Location	1; 1 cm										
<input type="checkbox"/> Size	2; 2 cm										
X	2 cm										
Y	2 cm										
<table border="1"> <tr><td><input type="checkbox"/> <b>Layout</b></td><td></td></tr> <tr><td><input checked="" type="checkbox"/> Location</td><td>1; 1 cm</td></tr> <tr><td><input type="checkbox"/> Size</td><td>1; 1 cm</td></tr> <tr><td>    X</td><td>1 cm</td></tr> <tr><td>    Y</td><td>1 cm</td></tr> </table>	<input type="checkbox"/> <b>Layout</b>		<input checked="" type="checkbox"/> Location	1; 1 cm	<input type="checkbox"/> Size	1; 1 cm	X	1 cm	Y	1 cm	
<input type="checkbox"/> <b>Layout</b>											
<input checked="" type="checkbox"/> Location	1; 1 cm										
<input type="checkbox"/> Size	1; 1 cm										
X	1 cm										
Y	1 cm										

Note 1: X coordinate for band equals 0 and does not vary.

Note 2: "Snap to Grid" option defines if the object snaps to the grid when its position changes or not. Use the appropriate button to turn on\off this option.

### CanBreak (Render)

Property allows the object's contents to break between pages and columns in case the object exceeds page size.

Example of a CanBreak property of Detail and AdvancedText element.

Property	Report												
<table border="1"> <tr><td><input type="checkbox"/> <b>Render</b></td><td></td></tr> <tr><td>CanBreak</td><td>False</td></tr> <tr><td>CanGrow</td><td><b>True</b></td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>GrowToBottom</td><td>False</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<input type="checkbox"/> <b>Render</b>		CanBreak	False	CanGrow	<b>True</b>	CanShrink	False	GrowToBottom	False	Visible	True	<p>dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad</p>
<input type="checkbox"/> <b>Render</b>													
CanBreak	False												
CanGrow	<b>True</b>												
CanShrink	False												
GrowToBottom	False												
Visible	True												
<table border="1"> <tr><td><input type="checkbox"/> <b>Render</b></td><td></td></tr> <tr><td>CanBreak</td><td><b>True</b></td></tr> <tr><td>CanGrow</td><td><b>True</b></td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>GrowToBottom</td><td>False</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<input type="checkbox"/> <b>Render</b>		CanBreak	<b>True</b>	CanGrow	<b>True</b>	CanShrink	False	GrowToBottom	False	Visible	True	<p>dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad</p> <p>minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in</p>
<input type="checkbox"/> <b>Render</b>													
CanBreak	<b>True</b>												
CanGrow	<b>True</b>												
CanShrink	False												
GrowToBottom	False												
Visible	True												





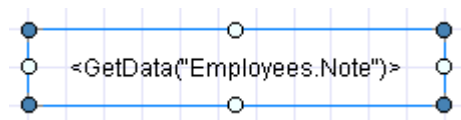
Note: when one sets a property CanBreak of the object that is located inside the other section, CanBreak property must be set for the container band in order for its content to break up into pages.

### CanGrow (Render)

Property allows increasing objects' height in case its contents do not fit size specified for the object in the template.

Example demonstrating use of CanGrow property of the TextBox element.

Report Template:



Property	Report										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>GrowToBottom</td><td>False</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	GrowToBottom	False	Visible	True	<div style="border: 1px solid blue; padding: 5px;">Education includes a BA in psychology from Colorado State Uni</div>
<b>Render</b>											
CanGrow	False										
CanShrink	False										
GrowToBottom	False										
Visible	True										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td><b>True</b></td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>GrowToBottom</td><td>False</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	<b>True</b>	CanShrink	False	GrowToBottom	False	Visible	True	<div style="border: 1px solid blue; padding: 5px;">Education includes a BA in psychology from Colorado State University in 1970. She also completed The Art of the Cold Call. Nancy is a member of Toastmasters International.</div>
<b>Render</b>											
CanGrow	<b>True</b>										
CanShrink	False										
GrowToBottom	False										
Visible	True										

Note: when CanGrow property is set for an object contained in another band, the same property must be set for the container band in order for it to change its height depending on the height of the contained objects.



### CanShrink (Render)

Property allows decreasing objects' height in case the content inserted during the report generation has smaller size than that set for the container.

Example demonstrating use of "CanShrink" property of the TextBox element.

Report Template:

```

<GetData
("Employee
s.Employee
eName")>

```

Property	Report										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>GrowToBottom</td><td>False</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	GrowToBottom	False	Visible	True	
<b>Render</b>											
CanGrow	False										
CanShrink	False										
GrowToBottom	False										
Visible	True										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td><b>True</b></td></tr> <tr><td>GrowToBottom</td><td>False</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	<b>True</b>	GrowToBottom	False	Visible	True	
<b>Render</b>											
CanGrow	False										
CanShrink	<b>True</b>										
GrowToBottom	False										
Visible	True										

Note: when CanShrink property is set for an object contained in another band, the same property must be set for the container band in order for it to change its height depending on the height of the contained objects.



### MinSpaceAfter (Render)

The property sets minimal space between lower border of the band and lower field of the page.

Sample of using MinSpaceAfter for the Detail band displaying list.

Report Template:

```
dataBand1:DataBand DataSource = Em
detail1:Detail
<dataBand1
("EmployeeName")>
end of dataBand1
```

Property		Report
CanGrow	False	
CanShrink	False	
MinSpaceAfter	<b>0,75 cm</b>	
NewColumnAfter	False	

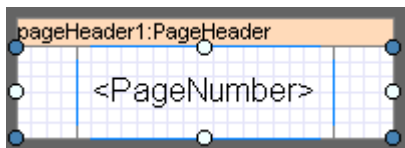


### Mode (Render)

Property allows setting headlines and overlay generation and defines elements display on even, odd or all the pages.

Example demonstrating use of Mode property of the PageHeader:

Report Template:



Property	Report													
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>Mode</td><td>AllPages</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	Mode	AllPages	Visible	True	<table border="1"> <tr><td>1</td></tr> <tr><td>2</td></tr> <tr><td>3</td></tr> </table>	1	2	3
<b>Render</b>														
CanGrow	False													
CanShrink	False													
Mode	AllPages													
Visible	True													
1														
2														
3														
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>Mode</td><td>OddPages</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	Mode	OddPages	Visible	True	<table border="1"> <tr><td>1</td></tr> <tr><td></td></tr> <tr><td>3</td></tr> </table>	1		3
<b>Render</b>														
CanGrow	False													
CanShrink	False													
Mode	OddPages													
Visible	True													
1														
3														
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanGrow</td><td>False</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>Mode</td><td>EvenPages</td></tr> <tr><td>Visible</td><td>True</td></tr> </table>	<b>Render</b>		CanGrow	False	CanShrink	False	Mode	EvenPages	Visible	True	<table border="1"> <tr><td></td></tr> <tr><td>2</td></tr> <tr><td></td></tr> </table>		2	
<b>Render</b>														
CanGrow	False													
CanShrink	False													
Mode	EvenPages													
Visible	True													
2														

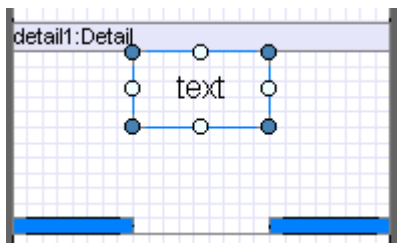


### GrowToBottom (Render)

Property allows the object to stretch up to the bottom border of the band it is contained in.

Example demonstrating use of TextBox.GrowToBottom property

Report Template:



Property	Report
<b>Render</b> CanGrow: False CanShrink: False GrowToBottom: False Visible: True	
<b>Render</b> CanGrow: False CanShrink: False GrowToBottom: <b>True</b> Visible: True	



### NewColumnAfter (Render)

Property sets if a new column is inserted after the band contents.

Example of adding a new column after the Detail contents:

Report Template:

```

dataBand3:DataBand DataSource = Em
detail3:Detail
  <dataBand3
  ("EmployeeNa
  me")>
end of dataBand3

```

Property		Report
<b>Render</b> CanBreak: False CanGrow: <b>True</b> CanShrink: <b>True</b> MinSpaceAfter: 0 cm NewColumnAfter: False NewColumnBefore: False		
<b>Render</b> CanBreak: False CanGrow: <b>True</b> CanShrink: <b>True</b> MinSpaceAfter: 0 cm NewColumnAfter: <b>True</b> NewColumnBefore: False		



### NewColumnBefore (Render)

Property sets if a new column is inserted before the band contents.

Example of adding a new column before the Detail contents (detail4):

Report Template:

```

dataBand3:DataBand DataSource = Em
detail4:Detail
<dataBand3("E
detail3:Detail
<dataBand3("H
end of dataBand3

```

Property	Report																										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanBreak</td><td>True</td></tr> <tr><td>CanGrow</td><td>True</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>MinSpaceAfter</td><td>0 cm</td></tr> <tr><td>NewColumnAfter</td><td>False</td></tr> <tr><td>NewColumnBefore</td><td>False</td></tr> </table>	<b>Render</b>		CanBreak	True	CanGrow	True	CanShrink	False	MinSpaceAfter	0 cm	NewColumnAfter	False	NewColumnBefore	False	<table border="1"> <tr><td>Nancy Davolio</td><td>(206) 555-9482</td></tr> <tr><td>(206)555-9857</td><td>Anne Dodsworth</td></tr> <tr><td>Andrew Fuller</td><td>(71) 555-4444</td></tr> <tr><td>Steven Buchana</td><td>(71) 555-4444</td></tr> <tr><td>(71) 555-4444</td><td>Anne Yorke</td></tr> <tr><td>Laura Callahan</td><td>(71) 555-4444</td></tr> </table>	Nancy Davolio	(206) 555-9482	(206)555-9857	Anne Dodsworth	Andrew Fuller	(71) 555-4444	Steven Buchana	(71) 555-4444	(71) 555-4444	Anne Yorke	Laura Callahan	(71) 555-4444
<b>Render</b>																											
CanBreak	True																										
CanGrow	True																										
CanShrink	False																										
MinSpaceAfter	0 cm																										
NewColumnAfter	False																										
NewColumnBefore	False																										
Nancy Davolio	(206) 555-9482																										
(206)555-9857	Anne Dodsworth																										
Andrew Fuller	(71) 555-4444																										
Steven Buchana	(71) 555-4444																										
(71) 555-4444	Anne Yorke																										
Laura Callahan	(71) 555-4444																										
<table border="1"> <tr><td colspan="2"><b>Render</b></td></tr> <tr><td>CanBreak</td><td>True</td></tr> <tr><td>CanGrow</td><td>True</td></tr> <tr><td>CanShrink</td><td>False</td></tr> <tr><td>MinSpaceAfter</td><td>0 cm</td></tr> <tr><td>NewColumnAfter</td><td>False</td></tr> <tr><td>NewColumnBefore</td><td>True</td></tr> </table>	<b>Render</b>		CanBreak	True	CanGrow	True	CanShrink	False	MinSpaceAfter	0 cm	NewColumnAfter	False	NewColumnBefore	True	<table border="1"> <tr><td>Nancy Davolio</td><td>Andrew Fuller</td></tr> <tr><td>(206)555-9857</td><td>(206) 555-9482</td></tr> <tr><td>Anne Dodsworth</td><td>Steven Buchana</td></tr> <tr><td>(71) 555-4444</td><td>(71) 555-4444</td></tr> </table>	Nancy Davolio	Andrew Fuller	(206)555-9857	(206) 555-9482	Anne Dodsworth	Steven Buchana	(71) 555-4444	(71) 555-4444				
<b>Render</b>																											
CanBreak	True																										
CanGrow	True																										
CanShrink	False																										
MinSpaceAfter	0 cm																										
NewColumnAfter	False																										
NewColumnBefore	True																										
Nancy Davolio	Andrew Fuller																										
(206)555-9857	(206) 555-9482																										
Anne Dodsworth	Steven Buchana																										
(71) 555-4444	(71) 555-4444																										



### NewPageAfter (Render)

Property sets if a new page is inserted after the band contents.

Example of adding a new page after the Detail contents:

Report Template:

```

dataBand3:DataBand DataSource = Em
detail3:Detail
  < dataBand3("EmployeeNa
end of dataBand3

```

Property		Report
MinSpaceAfter	0 cm	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Nancy Davolio Andrew Fuller</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Anne Dodsworth Steven Buchana</div>
NewColumnAfter	False	
NewColumnBefore	False	
NewPageAfter	False <input type="checkbox"/>	
NewPageBefore	False	
Visible	True	
MinSpaceAfter	0 cm	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Nancy Davolio</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Andrew Fuller</div>
NewColumnAfter	False	
NewColumnBefore	False	
NewPageAfter	<b>True</b> <input checked="" type="checkbox"/>	
NewPageBefore	False	
Visible	True	





### NewPageBefore (Render)

Property sets if a new page is inserted before the section contents.

Example of adding a new page before the Detail contents (detail3):

Report Template:

```

dataBand3:DataBand DataSource = Em
detail4:Detail
  <dataBand3("EmployeeName")>
detail3:Detail
  <dataBand3("HomePhone")>
end of dataBand3

```

Property		Report
MinSpaceAfter	0 cm	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Nancy Davolio (206)555-9857 Andrew Fuller</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">(206) 555-9482 Anne Dodsworth (71) 555-4444</div>
NewColumnAfter	False	
NewColumnBefore	False	
NewPageAfter	False	
NewPageBefore	False <input type="checkbox"/>	
Visible	True	
MinSpaceAfter	0 cm	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Nancy Davolio</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">(206)555-9857 Andrew Fuller</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">(206) 555-9482 Anne Dodsworth</div>
NewColumnAfter	False	
NewColumnBefore	False	
NewPageAfter	False	
NewPageBefore	<b>True</b> <input checked="" type="checkbox"/>	
Visible	True	

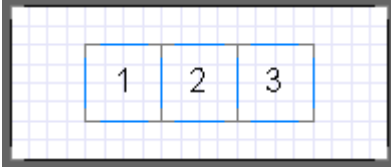


### Visible (Render)

Property sets visibility of the object in the report.

Example of visibility property of Textbox element with "2" digit:

Report Template:



Property	Report										
<table border="1"><tr><td colspan="2"><input checked="" type="checkbox"/> <b>Render</b></td></tr><tr><td>CanGrow</td><td>False</td></tr><tr><td>CanShrink</td><td>False</td></tr><tr><td>GrowToBottom</td><td>False</td></tr><tr><td>Visible</td><td>True <input type="checkbox"/></td></tr></table>	<input checked="" type="checkbox"/> <b>Render</b>		CanGrow	False	CanShrink	False	GrowToBottom	False	Visible	True <input type="checkbox"/>	
<input checked="" type="checkbox"/> <b>Render</b>											
CanGrow	False										
CanShrink	False										
GrowToBottom	False										
Visible	True <input type="checkbox"/>										
<table border="1"><tr><td colspan="2"><input checked="" type="checkbox"/> <b>Render</b></td></tr><tr><td>CanGrow</td><td>False</td></tr><tr><td>CanShrink</td><td>False</td></tr><tr><td>GrowToBottom</td><td>False</td></tr><tr><td>Visible</td><td><b>False</b> <input type="checkbox"/></td></tr></table>	<input checked="" type="checkbox"/> <b>Render</b>		CanGrow	False	CanShrink	False	GrowToBottom	False	Visible	<b>False</b> <input type="checkbox"/>	
<input checked="" type="checkbox"/> <b>Render</b>											
CanGrow	False										
CanShrink	False										
GrowToBottom	False										
Visible	<b>False</b> <input type="checkbox"/>										

Note: if a container section is invisible, all the containing objects are not displayed also.



### Text Output

The following elements are used in SharpShooter Reports for text input:

Elements	Descriptions
TextBox	Element allows input of simple text information and has text displaying adjustments.
AdvancedText	Element outputs formatted text
RichText	Element to output RTF text

### Static Text

Text property is used for text input that is output in the TextBox element. This property is defined in Text Editor. Double-click on TextBox or press the button of the Text property on the Property grid to open the editor. One can also type the text straight into the Text cell.

Appearance	
Angle	0
Border	
Fill	[None]
Font	Arial; 12pt
Margins	0; 0; 0; 0 cm
RightToLeft	False
StringTrimming	Character
StyleName	
Text	<b>text</b>
TextAlign	MiddleCenter
TextFill	[None]
TextFormat	
ToolTip	

### Text from the Data Source

Set the Value property of the TextBox element that gets the value of the selected field in order to input text from the data source. One can use GetData function or use DataBand indexer.

(More information about lists building)

### Border, Text in Cells

Border property is used for defining element borders. One can input information as a table, e.g.:

Caption		
Column	Cell#1	Cell#2
	Cell#3	



### Text Output LTR

Text output direction can be changed for the text output in TextBox and AdvancedText elements. Set *RightToLeft* property to True to do it.

Text property = "7 8 9 10 11"

Property		Report
RightToLeft	False	7 8 9 10 11
StringTrimming	Character	
StyleName		
Text	7 8 9 10 11	
RightToLeft	True	11 10 9 8 7
StringTrimming	Character	
StyleName		
Text	7 8 9 10 11	

### Text Cutting

One can set text cutting settings using the *StringTrimming* property in case output text does not fit the element area.

Property		Report
Text is cut after the last visible symbol:		
RightToLeft	False	one two t
StringTrimming	Character	
StyleName		
Text	one two three	
Text is cut after the last visible word:		
RightToLeft	False	one two
StringTrimming	Word	
StyleName		
Text	one two three	
Text is cut after the last visible symbol and the ellipsis is added:		
RightToLeft	False	one tw...
StringTrimming	EllipsisCharacter	
StyleName		
Text	one two three	
Text is cut after the last visible word and the ellipsis is added:		
RightToLeft	False	one ...
StringTrimming	EllipsisWord	
StyleName		
Text	one two three	
The text beginning and the ending is displayed, ellipsis is inserted in between:		
RightToLeft	False	on...ree
StringTrimming	EllipsisWord	
StyleName		
Text	one two three	



*ClipText* property defines how to handle text lines that don't entirely fit into TextBox element area.

Property		Report
Border	<b>Border</b>	
ClipText	False	
Fill	[None]	
Font	<b>Calibri; 39,75pt; Bold=Off; It</b>	
Border	<b>Border</b>	
ClipText	<b>True</b>	
Fill	[None]	
Font	<b>Calibri; 39,75pt; Bold=Off; It</b>	

### Managing Text Element Size

Size property defines the size of an element. There is a feature to change element size right during the generation process. It is useful when element content is not available while the template is being created or the content can vary during report generation. *CanShrink\_* and *CanGrow* properties allow the element to grow (or does not allow, depending on the value) according to the contents. *GrowToBottom* property defines element size increase up to the bottom border of the containing element.

### Text Style

There is a wide choice of instruments for report creation. Fill property defines the background color. Font property forms text features such as font type, text styles (normal, bold, italic, underlined and strikethrough). Text's align and position inside the element are defined with *TextAlign* property. One can set style and text background color using *TextFill* property.

Also there is an opportunity to apply one style to several elements. Style Sheet Editor is used as a style editor *StyleName* applies style to an element.

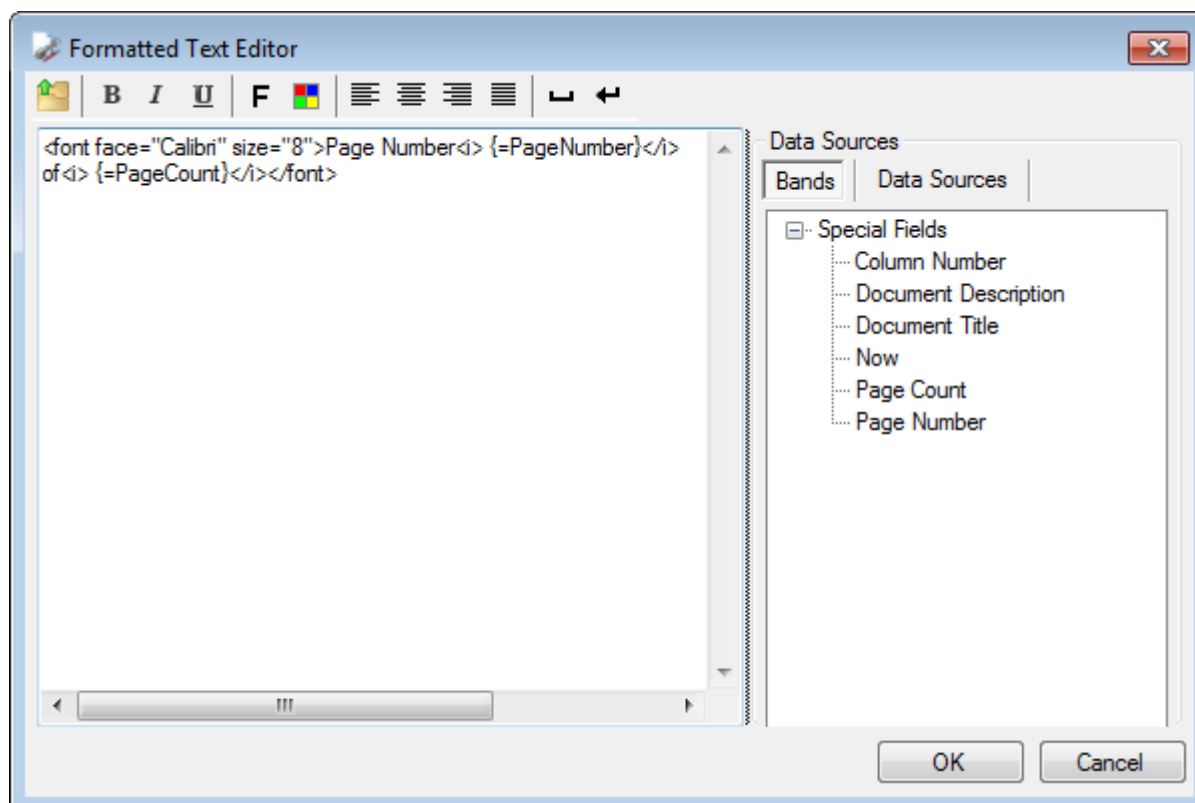
### Text Format

TextFormat property can be used to display text in a special format. There are several text formats available in Format editor: general, number, currency, date and time, percentage. User format is also available.

### Formatted Text

AdvancedText component allows you to input information as a formatted text. There is an option of paragraph and text styles setting and statement using right in the marked text. Formatted text can be specified through HTML-like markup language (*Text* property) and RTF format subset (*RTF* property).

Text can be formatted through HTML-like tags in Formatted Text Editor (*Text* property)



### Paragraph Formatting

The couple of tags `<P>` and `</P>` forms selected text as a separate paragraph.

Align attribute specifies horizontal alignment of the text in a paragraph. Available values:

align = "left" – alignment to the left margin of the paragraph

align = "right" - alignment to the right margin of the paragraph

align = "center" – center text

align = "justify" – align text to both left and right margins

Default value of alignment is "left". Select edited text and use "Left Align", "Center Align", "Right Align" and "Justify" buttons on the toolbox to set the alignment.

### Text Formatting

`<Font>` tag sets the output style of the following text and the closing tag `</font>` cancels previous settings. `<font>` tags enclosure is available. Text style is specified through the following attributes:



face – sets font name,

size – sets font size.

Press “Font” button to open style editor for quick style definition.

color – sets font color.

Available values: color name (e.g. color=“red”), hexadecimal format of separate color components in RGB (e.g. color=“#FF0000”). Press “Color” button to open ColorPicker for quick color choice.

Tag <B> - the following text will be of a bold type. Closing tag </B> cancels this setting. (“Bold” button).

Tag <I> - the following text will be of an italic type. Closing tag </I> cancels this setting. (“Italic” button)

Tag <U> - the following text will be underlined. Closing tag </U> cancels this setting. (“Underline” button)

Tag <BR> - lines folding in a paragraph. (“Insert line break” button).

### Symbol Insert

“&nbsp;” – space (“Insert space” button),

“&” – ‘&’ symbol,

“&lt;” - ‘<’ symbol,

“&#ddd;” - symbol with ddd code.

Note: If the text is not marked as a paragraph, alignment settings according to the TextAlign property are applied. If there are no evident font settings, the settings according to Font property are applied. If text color is not specified, TextFill property is applied. The following is ignored during the text handling: line breaks, several spaces, in succession and unknown tags and attributes. Case in tag names doesn’t matter.

### Using Expressions

One can use statements (the result of those will be inserted in the text) directly in the marked text. It is possible to use expressions which calculation result will be set into the text can be used directly inside marked up text. The expression specifies as {=<Statement>}. {=<Statement>} – expression on the set script language. All the available types and objects can be used in the expression. Formatting according to the mask specified in the Format property is applied to the expression result before it is inserted in the text. In case an error occurs, error message is inserted in the text as an expression result.

DataSources section adds special properties to the documents and Data Source fields.

### Using RTF Property

RTF format subset is used for text formatting. The following constructions from the RTF format are used: font table, color table, comments, paragraph format operators (“\par”, “\pard”, “\ql”, “\qr”, “\qc”, “\qj”, “\line”), text format operators (“\fxx”, “\fsxx”, “\cfxx”, “\b”, “\i”, “\u”). All the rest constructions and operators will be ignored. Rtf property is designed to write only. Rtf property is connected to the Text property: when Rtf property is specified, Text property value specifies automatically.

Press “Open RTF document” button to load text from the RTF file.



### *Breaking Text between Pages*

AdvancedText element allows to output text from one element to one or more pages. CanBreak property is used for it.

### *RTF Text*

RichText element is used for RTF-text output. Text loads from file through RichText Editor.





## List

### Simple List

DataBand is used for lists building. The band repeatedly outputs elements embedded into it. Visual elements output as a list must be contained in the Detail inside DataBand. Detail sets row height. InstanceCount property sets the amount of repetitions.

<b>Data</b>	
DataSource	
FilterExpression	
InstanceCount	5
Sort	(Collection)
Tag	
Totals	(Collection)

<DataBand name>.LineNumber() statement is used for getting line number.

Example of a list that outputs line number in the list:

```
dataBand1.InstanceCount = 5;
```

```
textBox.Value = "line number = " + dataBand1.LineNumber().ToString() + " of " + dataBand1.InstanceCount.ToString();
```

Report Template	Report
<pre>dataBand1:DataBand detail1:Detail   &lt;"line number: " + dataBa end of dataBand1</pre>	<pre>line number: 1 of 5 line number: 2 of 5 line number: 3 of 5 line number: 4 of 5 line number: 5 of 5</pre>
<pre>dataBand1:DataBand detail1:Detail   &lt;"line number: " + dataBa end of dataBand1</pre>	<pre>line number: 1 of 5 line number: 2 of 5 line number: 3 of 5</pre>

### List from the Data Source

DataBand allows building a list based on the data from the source. Data source is set in DataSource property. Note: the number of output lines depends on the amount of lines in the data source, ignoring InstanceCount property.

<b>Data</b>	
DataSource	Employees
FilterExpression	
InstanceCount	0
Sort	(Collection)
Tag	
Totals	(Collection)

In order to get access to the current element that has to be output, either GetData("<Data source name>.<Field name>") or <DataBand name>["<Field name>"] script is used. Textual information can be output by dragging the corresponding field from the document tree on the Data Sources tab into Detail (located in DataBand).

Example of a list that outputs employees' names:



```
dataBand.DataSource = Employees;
textBox.Value = dataBand["EmployeeName"];
```

Report Template	Report
<pre>dataBand:DataBand DataSource = Emp detail:Detail &lt;dataBand["EmployeeNa end of dataBand</pre>	<pre>Nancy Davolio Andrew Fuller Anne Dodsworth Steven Buchana Laura Callahan</pre>

### List in Columns

To save report space and make report more visual, there is an option to output the list as a set number of columns. The number is set in the ColumnsCount property of the DataBand. One can also set the indent between the columns in the list, ColumnsGap property serves for it.

Example of the employee list:

```
dataBand.DataSource = Employees;
textBox.Value = dataBand["EmployeeName"];
```

Report Template:

dataBand:DataBand DataSource = Emp
detail:Detail
<dataBand["
end of dataBand

Property	Report										
<table border="1"> <thead> <tr> <th colspan="2">Layout</th> </tr> </thead> <tbody> <tr> <td>ColumnsCount</td> <td>0</td> </tr> <tr> <td>ColumnsGap</td> <td>0 cm</td> </tr> <tr> <td>Location</td> <td>0; 0 cm</td> </tr> <tr> <td>Size</td> <td>5; 2 cm</td> </tr> </tbody> </table>	Layout		ColumnsCount	0	ColumnsGap	0 cm	Location	0; 0 cm	Size	5; 2 cm	<pre>Nancy Davolio Andrew Fuller Anne Dodsworth  Steven Buchana Laura Callahan Anne Yorke</pre>
Layout											
ColumnsCount	0										
ColumnsGap	0 cm										
Location	0; 0 cm										
Size	5; 2 cm										
<table border="1"> <thead> <tr> <th colspan="2">Layout</th> </tr> </thead> <tbody> <tr> <td>ColumnsCount</td> <td>2</td> </tr> <tr> <td>ColumnsGap</td> <td>0 cm</td> </tr> <tr> <td>Location</td> <td>0; 0 cm</td> </tr> <tr> <td>Size</td> <td>6; 1.6 cm</td> </tr> </tbody> </table>	Layout		ColumnsCount	2	ColumnsGap	0 cm	Location	0; 0 cm	Size	6; 1.6 cm	<pre>Nancy          Steven Davolio        Buchana Andrew         Laura Fuller         Callahan Anne           Anne Yorke Dodsworth</pre>
Layout											
ColumnsCount	2										
ColumnsGap	0 cm										
Location	0; 0 cm										
Size	6; 1.6 cm										
<table border="1"> <thead> <tr> <th colspan="2">Layout</th> </tr> </thead> <tbody> <tr> <td>ColumnsCount</td> <td>2</td> </tr> <tr> <td>ColumnsGap</td> <td>0,25 cm</td> </tr> <tr> <td>Location</td> <td>0; 0 cm</td> </tr> <tr> <td>Size</td> <td>5; 1,5 cm</td> </tr> </tbody> </table>	Layout		ColumnsCount	2	ColumnsGap	0,25 cm	Location	0; 0 cm	Size	5; 1,5 cm	<pre>Nancy          Steven Davolio        Buchana Andrew         Laura Fuller         Callahan Anne           Anne Yorke Dodsworth</pre>
Layout											
ColumnsCount	2										
ColumnsGap	0,25 cm										
Location	0; 0 cm										
Size	5; 1,5 cm										

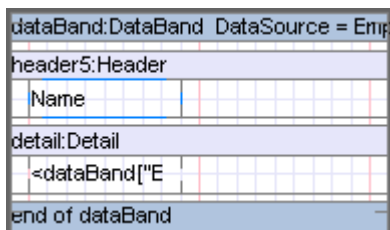
### List Header

Headers are used for data output once in the beginning of a list, page or column. Header is used for this purpose and it must be located inside DataBand. RepeatEveryColumn and RepeatEveryPage properties serve for the heading repetition mode.

Example of the list heading usage:

```
dataBand3.DataSource = Employees; textBox1.Text = "Name"; textBox2.Value = dataBand3("EmployeeName");
```

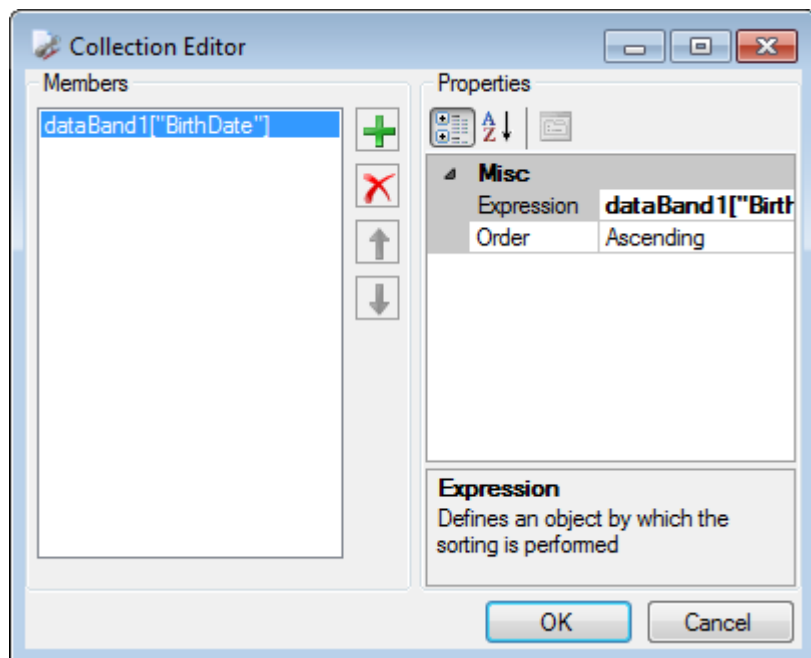
Report Template:



Property		Report
NewColumnBefore	False	
NewPageAfter	False	
NewPageBefore	False	
RepeatEveryColumn	False	
RepeatEveryPage	False	
Visible	True	
NewColumnBefore	False	
NewPageAfter	False	
NewPageBefore	False	
RepeatEveryColumn	<b>True</b>	
RepeatEveryPage	False	
Visible	True	
NewPageAfter	False	
NewPageBefore	False	
RepeatEveryColumn	False	
RepeatEveryPage	False	
Visible	True	
NewPageAfter	False	
NewPageBefore	False	
RepeatEveryColumn	False	
RepeatEveryPage	<b>True</b>	
Visible	True	

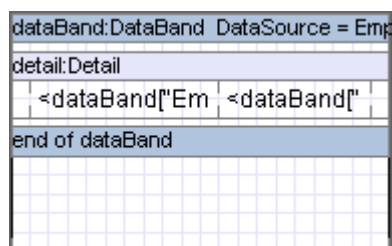
### List Sorting

DataBand outputs entries in the same order they are saved in the data source. Sort property of DataBand sets the special output order. Collection Editor is used for the property definition. Expression property is used for setting the expression of the sorting (the script must be used as an expression); Order property defines the order of the sorting (ascending or descending).



Example of a list that outputs names and birthdates of the employees according to the sorting condition:

Report Template:



```
dataBand.DataSource = Employees;
```

```
textBoxName.Value = dataBand["EmployeeName"];
```

```
textBoxDate.Value = dataBand["BirthDate"];
```



Sorting conditions	Report															
None	<table border="1"> <tr><td>Nancy Davolio</td><td>11.11.1976</td></tr> <tr><td>Andrew Fuller</td><td>25.02.1969</td></tr> <tr><td>Anne Dodswort</td><td>30.06.1981</td></tr> <tr><td>Steven Buchan</td><td>15.12.1984</td></tr> <tr><td>Laura Callahan</td><td>28.09.1987</td></tr> </table>	Nancy Davolio	11.11.1976	Andrew Fuller	25.02.1969	Anne Dodswort	30.06.1981	Steven Buchan	15.12.1984	Laura Callahan	28.09.1987					
Nancy Davolio	11.11.1976															
Andrew Fuller	25.02.1969															
Anne Dodswort	30.06.1981															
Steven Buchan	15.12.1984															
Laura Callahan	28.09.1987															
<table border="1"> <tr><td><input checked="" type="checkbox"/> Misc</td></tr> <tr><td>Expression</td><td>dataBand["EmployeeName"]</td></tr> <tr><td>Order</td><td>Ascending</td></tr> </table>	<input checked="" type="checkbox"/> Misc	Expression	dataBand["EmployeeName"]	Order	Ascending	<table border="1"> <tr><td>Andrew Fuller</td><td>25.02.1969</td></tr> <tr><td>Anne Dodswort</td><td>30.06.1981</td></tr> <tr><td>Anne Yorke</td><td>03.05.1969</td></tr> <tr><td>Laura Callahan</td><td>28.09.1987</td></tr> <tr><td>Nancy Davolio</td><td>11.11.1976</td></tr> </table>	Andrew Fuller	25.02.1969	Anne Dodswort	30.06.1981	Anne Yorke	03.05.1969	Laura Callahan	28.09.1987	Nancy Davolio	11.11.1976
<input checked="" type="checkbox"/> Misc																
Expression	dataBand["EmployeeName"]															
Order	Ascending															
Andrew Fuller	25.02.1969															
Anne Dodswort	30.06.1981															
Anne Yorke	03.05.1969															
Laura Callahan	28.09.1987															
Nancy Davolio	11.11.1976															
<table border="1"> <tr><td><input checked="" type="checkbox"/> Misc</td></tr> <tr><td>Expression</td><td>dataBand["BirthDate"]</td></tr> <tr><td>Order</td><td>Descending</td></tr> </table>	<input checked="" type="checkbox"/> Misc	Expression	dataBand["BirthDate"]	Order	Descending	<table border="1"> <tr><td>Laura Callahan</td><td>28.09.1987</td></tr> <tr><td>Steven Buchan</td><td>15.12.1984</td></tr> <tr><td>Anne Dodswort</td><td>30.06.1981</td></tr> <tr><td>Nancy Davolio</td><td>11.11.1976</td></tr> <tr><td>Anne Yorke</td><td>03.05.1969</td></tr> </table>	Laura Callahan	28.09.1987	Steven Buchan	15.12.1984	Anne Dodswort	30.06.1981	Nancy Davolio	11.11.1976	Anne Yorke	03.05.1969
<input checked="" type="checkbox"/> Misc																
Expression	dataBand["BirthDate"]															
Order	Descending															
Laura Callahan	28.09.1987															
Steven Buchan	15.12.1984															
Anne Dodswort	30.06.1981															
Nancy Davolio	11.11.1976															
Anne Yorke	03.05.1969															

### List Filtering

In case it is necessary to display the information that fulfills some conditions, one can use FilterExpression property of the DataBand. The condition is to be specified in the Script Editor.

Example of the list that displays employees' names according to the sex

Report Template:

dataBand:DataBand DataSource = Emp
detail:Detail
<dataBand["EmployeeNa
end of dataBand

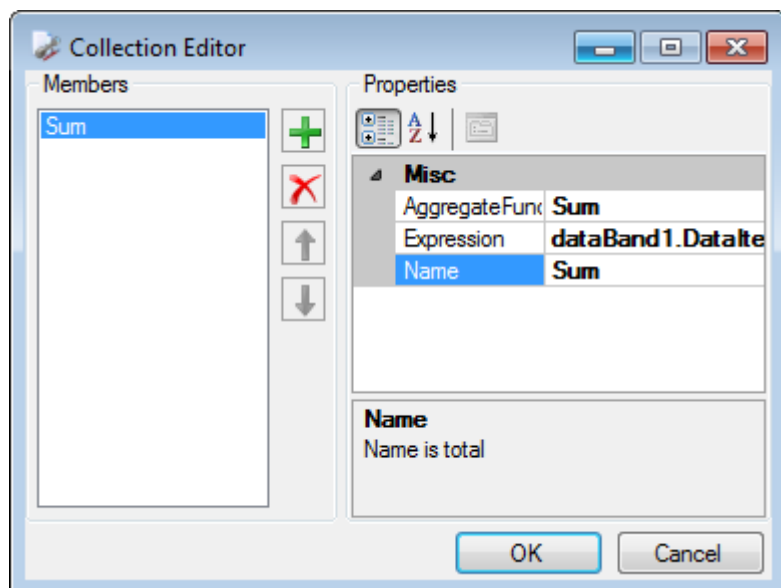
```
dataBand.DataSource = Employees;
```

```
textBoxName.Value = dataBand["EmployeeName"];
```

Property	Report				
<pre>dataBand.FilterExpression = dataBand1["Sex"].ToString() == "f"</pre>	<table border="1"> <tr><td>Nancy Davolio</td></tr> <tr><td>Anne Dodsworth</td></tr> <tr><td>Laura Callahan</td></tr> <tr><td>Anne Yorke</td></tr> </table>	Nancy Davolio	Anne Dodsworth	Laura Callahan	Anne Yorke
Nancy Davolio					
Anne Dodsworth					
Laura Callahan					
Anne Yorke					
<pre>dataBand.FilterExpression = dataBand1["Sex"].ToString() == "m"</pre>	<table border="1"> <tr><td>Andrew Fuller</td></tr> <tr><td>Steven Buchana</td></tr> </table>	Andrew Fuller	Steven Buchana		
Andrew Fuller					
Steven Buchana					

### Final List Values

One can use aggregated lists when it is necessary to count the totals of the list. The list of the aggregate functions for the data output in the DataBand can be specified in the Totals property with the help of Collection Editor.



AggregateFunction property specifies the aggregated function. The aggregated expression is set by the Expression property. The name of the function is to be specified in the Name property. GetTotal("<aggregate name>") script is used for calling the function and getting the sum; GetRunningTotal("<aggregate name>") is used in order to get current cumulative value. One can get arroagated data before the output of the aggregated values. GetTotal result does not depend on the location of the caller element: before or after. It is recommended to output GetRunningTotal value in the Detail; calling from the Header is useless because there are no accumulated values; GetRunningTotal value called from the Footer will coincide with GetTotal value.

Report generator supports 12 built-in aggregate functions.

Example of the aggregate functions operating:

Report Template:

```
dataBand:DataE
detail:Detail
<dataBa
footer:Footer
<GetTota
end of dataBand
```

```
dataBand.DataSource = Numbers; textBox1.Value = dataBand["Value"]; textBox.Value =
GetTotal("Func");
```

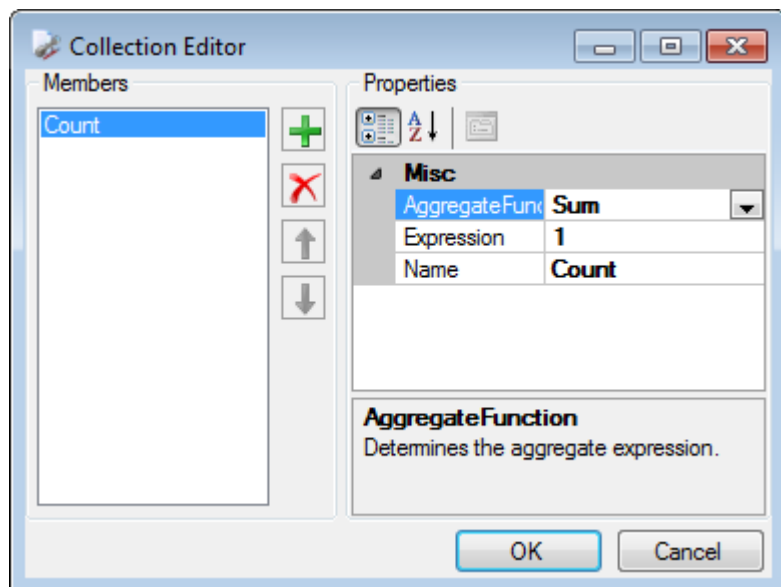


Property	Report												
<b>Total:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Sum</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Sum</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>91</td></tr> </table>	1	13	77	91
<b>Misc</b>													
AggregateFunction	<b>Sum</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
91													
<b>Arithmetic mean:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Avg</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Avg</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>30,33333</td></tr> </table>	1	13	77	30,33333
<b>Misc</b>													
AggregateFunction	<b>Avg</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
30,33333													
<b>Minimal element:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Min</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Min</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>1</td></tr> </table>	1	13	77	1
<b>Misc</b>													
AggregateFunction	<b>Min</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
1													
<b>Maximal element:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Max</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Max</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>77</td></tr> </table>	1	13	77	77
<b>Misc</b>													
AggregateFunction	<b>Max</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
77													
<b>Elements amount:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Count</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Count</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>3</td></tr> </table>	1	13	77	3
<b>Misc</b>													
AggregateFunction	<b>Count</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
3													
<b>Standard Deviation:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Standard Deviatio</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Standard Deviatio</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>33,35998</td></tr> </table>	1	13	77	33,35998
<b>Misc</b>													
AggregateFunction	<b>Standard Deviatio</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
33,35998													
<b>Estimate of Standard Deviation:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Estimate of Standa</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Estimate of Standa</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>40,85747</td></tr> </table>	1	13	77	40,85747
<b>Misc</b>													
AggregateFunction	<b>Estimate of Standa</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
40,85747													
<b>Variance:</b>													
<table border="1"> <tr> <td colspan="2"><b>Misc</b></td> </tr> <tr> <td>AggregateFunction</td> <td><b>Variance</b></td> </tr> <tr> <td>Expression</td> <td><b>dataBand["Value"]</b></td> </tr> <tr> <td>Name</td> <td><b>Func</b></td> </tr> </table>	<b>Misc</b>		AggregateFunction	<b>Variance</b>	Expression	<b>dataBand["Value"]</b>	Name	<b>Func</b>	<table border="1"> <tr><td>1</td></tr> <tr><td>13</td></tr> <tr><td>77</td></tr> <tr><td>1112,888</td></tr> </table>	1	13	77	1112,888
<b>Misc</b>													
AggregateFunction	<b>Variance</b>												
Expression	<b>dataBand["Value"]</b>												
Name	<b>Func</b>												
1													
13													
77													
1112,888													
<b>Estimate of variance:</b>													

<div style="border: 1px solid gray; padding: 2px;"> <input type="checkbox"/> <b>Misc</b>            AggregateFunction <b>Estimate of varian</b> ▾            Expression <b>dataBand["Value"]</b>            Name <b>Func</b> </div>		<div style="border: 1px solid gray; padding: 2px;">             1 13 77  <span style="border: 1px solid blue; padding: 2px;">1669,333</span> </div>
Median:		
<div style="border: 1px solid gray; padding: 2px;"> <input type="checkbox"/> <b>Misc</b>            AggregateFunction <b>Median</b> ▾            Expression <b>dataBand["Value"]</b>            Name <b>Func</b> </div>		<div style="border: 1px solid gray; padding: 2px;">             1 13 77  <span style="border: 1px solid blue; padding: 2px;">13</span> </div>
Skewness :		
<div style="border: 1px solid gray; padding: 2px;"> <input type="checkbox"/> <b>Misc</b>            AggregateFunction <b>Skewness</b> ▾            Expression <b>dataBand["Value"]</b>            Name <b>Func</b> </div>		<div style="border: 1px solid gray; padding: 2px;">             1 13 77  <span style="border: 1px solid blue; padding: 2px;">0,639105</span> </div>
Kurtosis:		
<div style="border: 1px solid gray; padding: 2px;"> <input type="checkbox"/> <b>Misc</b>            AggregateFunction <b>Kurtosis</b> ▾            Expression <b>dataBand["Value"]</b>            Name <b>Func</b> </div>		<div style="border: 1px solid gray; padding: 2px;">             1 13 77  <span style="border: 1px solid blue; padding: 2px;">-1,5</span> </div>

### Entries Amount on the Page

Total property of the DataBand can be used in order to count the entries located on the page. One can define count function in the Collection Editor in the following way:



PageFooter including Textbox element is used in order to output entries amount. Text property is set in PageFooter.GenerateScript:

```

if (Engine.IsDoublePass)
{
    textBoxFooter.Text = "Records " + dataBand5.GetPageTotal("Count") + " of "
+ dataBand5.GetTotal("Count");
}

```

One should set DoublePass property of the document to True in order to calculate entries.





Report Template	Report
<pre>dataBand5:DataBand DataSource = Em detail6:Detail &lt;dataBand5["EmployeeNam end of dataBand5 pageFooter1:PageFooter</pre>	<p>Nancy Davolio Andrew Fuller Anne Dodsworth Steven Buchana</p> <hr/> <p>Records 4 of 6</p>
	<p>Laura Callahan Anne Yorke</p> <hr/> <p>Records 2 of 6</p>



## Grouping

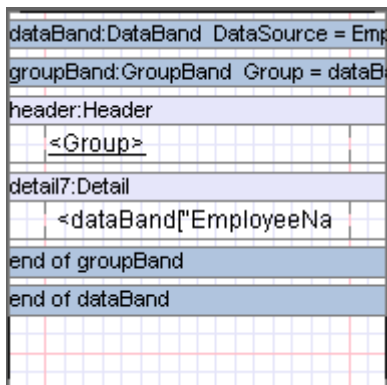
Data grouping according to the conditions can be used for a logic division of the lines with data and better information visualization. GroupBand is used for this. Grouping criterion is set by the GroupExpression property that is set in the Script Editor in turn.

### Group Headers

Put Header into GroupBand to display group header. Use Group expression to display grouping value.

Example of an employee list grouped by the first letter of a name

Report Template:



```
dataBand.DataSource = Employees; groupBand1.GroupExpression =
dataBand6["EmployeeName"].ToString().Substring(0,1); textBoxHeader.Value = Group;
textBoxDetail.Value = dataBand["EmployeeName"]
```

Report:

<u>N</u> Nancy Davolio
<u>A</u> Andrew Fuller Anne Dodsworth Anne Yorke
<u>S</u> Steven Buchana
<u>L</u> Laura Callahan



### Sorting in Groups

GroupBand outputs the entries into the report in the same order they are saved in the data source. Order property of the GroupBand sets the output order. There are 3 values of this property available: KeepOriginal – initial entries order, Ascending – the ascending sort of the entries, Descending – the descending sort of the entries.

<b>Data</b>	
GroupExpression	dataBand["EmployeeName"]
Order	Ascending
SortExpression	
Tag	

SortExpression property sets sorting criterion.

Example of an ordered employee list grouped by the first letter of the name:

```
dataBand.DataSource = Employees; groupBand.GroupExpression =
dataBand["EmployeeName"].ToString().Substring(0,1); groupBand.Order = Ascending;
textBoxHeader.Value = Group; textBoxDetail.Value = dataBand["EmployeeName"]
```

Report Template	Report
dataBand:DataBand DataSource = Emp	
groupBand:GroupBand Group = dataBe	<u>A</u>
header:Header	Andrew Fuller
<Group>	Anne Dodsworth
detail7:Detail	Anne Yorke
<dataBand["EmployeeNa	<u>L</u>
end of groupBand	Laura Callahan
end of dataBand	<u>N</u>
	Nancy Davolio
	<u>S</u>
	Steven Buchana



### Cascading Grouping

There is an option of creating a report with a complex structure of grouping. Several GroupBands nested into each other bands are used for this.

Example of an employee list grouped by countries first and then by cities:

```
dataBand.DataSource = Employees; groupBandCounty.GroupExpression = dataBand["Country"];
groupBandCity.GroupExpression = dataBand["City"]; textBoxDetail.Value =
dataBand["EmployeeName"].
```

Report Template	Report
dataBand:DataBand DataSource = Employees	
groupBandCounty:GroupBand Group = dataBand["Country"]	<u>Italy</u>
header1:Header	<u>Rome</u>
<Group>	Nancy Davolio
groupBandCity:GroupBand Group = dataBand["City"]	<u>Milan</u>
header2:Header	Anne Dodswort
<Group>	<u>France</u>
detail8:Detail	<u>Paris</u>
<dataBand["Em	Andrew Fuller
end of groupBandCity	
end of groupBandCounty	
end of dataBand	<u>England</u>
	<u>London</u>
	Steven Buchan
	<u>Liverpool</u>
	Laura Callahan
	Anne Yorke



### Group Totals

One should call GetTotal inside a group to get total value result of a group. Calling GetRunningTotal inside a group gets accumulated result in the group.

Example of a grouped numbers list, every group's sum is counted down first, at the end the total result of all the groups.

```
dataBand1.DataSource = Numbers;
```

```
Aggregate function: AggregateFunction = Sum; Expression = dataBand1 ["Value"]; Name = SUM;
```

```
groupBand4.GroupExpression = dataBand1["Type"]; textBoxDetail.Value = dataBand1["Value"];  
textBoxFooter.Value = GetTotal("SUM");
```

Report Template	Report
dataBand1:DataBand DataSource	1
groupBand4:GroupBand Group	13
detail9:Detail	12
<data	group sum 26
footer2:Footer	-34
group sum <Get	-56
end of groupBand4	108
footer3:Footer	group sum 18
total sum <Get	total sum 44
end of dataBand1	



## Horizontal List and Table

### Horizontal List

CrossBand is used for horizontal lists building. Band repeatedly outputs included elements LTR. CrossBand must be located in Detail, Header or Footers that define CrossBand location in a report. Repetition amount is defined by the InstanceCount property.

<b>Data</b>	
DataSource	
InstanceCount	5
Tag	

<CrossBand name>. LineNumber() statement is used in order to get entry number. If all the entries output by CrossBand don't fit into one line, the list continues from a new page.

Example of a horizontal list that outputs an entry number:

```
crossBand1.InstanceCount = 4;
textBox.Value = crossBand1.LineNumber().ToString + " of " +
crossBand1.InstanceCount().ToString;
```

Report Template	Report
<pre> &lt;CrossBand   &lt;cross   &lt;/CrossBand </pre>	
<pre> &lt;CrossBand1   &lt;cross   &lt;/CrossBand </pre>	

### Horizontal List from the Data Source

CrossBand allows building a horizontal list based on the data from the data source. Data source is set in the DataSource property; amount of output entries depends on a number of entries in a source and InstanceCount property is ignored.

<b>Data</b>	
DataSource	Numbers
InstanceCount	0
Tag	

GetData("<Data source name>.<Field name>") script is used in order to get access to the current element that is to be output.

Example of a horizontal list that outputs a row of numbers:  
crossBand1.DataSource = Numbers; textBox.Value = GetData("Numbers.Value")

Report Template	Report
<pre> &lt;CrossBand1   &lt;Get   &lt;/CrossBand </pre>	



### Table

Using CrossBand included into DataBand one can output data as a table.

Example of a calendar output:

```
dataBand.InstanceCount = 5; crossBand. InstanceCount = 7; textBox.Value =  
(crossBand.LineNumber + ((dataBand.LineNumber - 1) * 7))>31?"":(crossBand.LineNumber +  
((dataBand.LineNumber - 1) * 7)).ToString();
```

Report Template	Report																																																	
<pre>dataBand:DataBand header:Header     March, 2010     Sun Mon Tue Wed Thu Fri Sat dataBand CrossBand &lt;(cr end of dataBand R</pre>	<table border="1"><thead><tr><th colspan="7">March, 2010</th></tr><tr><th>Sun</th><th>Mon</th><th>Tue</th><th>Wed</th><th>Thu</th><th>Fri</th><th>Sat</th></tr></thead><tbody><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td></tr><tr><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td></tr><tr><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td></tr><tr><td>29</td><td>30</td><td>31</td><td></td><td></td><td></td><td></td></tr></tbody></table>	March, 2010							Sun	Mon	Tue	Wed	Thu	Fri	Sat	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
March, 2010																																																		
Sun	Mon	Tue	Wed	Thu	Fri	Sat																																												
1	2	3	4	5	6	7																																												
8	9	10	11	12	13	14																																												
15	16	17	18	19	20	21																																												
22	23	24	25	26	27	28																																												
29	30	31																																																



### SideBySide Bands

The SideBySide band is used for horizontal SideBySide output of the elements from different bands. All the elements, located into it, are output from the beginning of this band. The BandContainer element is used for merging several sections together.

Example of SideBySide output from the different bands:

dataBand1.InstanceCount = 5; dataBand2.InstanceCount = 4; dataBand3.InstanceCount = 2;

Report Template	Template Structure	Report
<pre> sideBySide1:SideBySide dataBand1:DataBand   detail1:Detail     1   end of dataBand1 dataBand2:DataBand   detail2:Detail     2   end of dataBand2 dataBand3:DataBand   detail3:Detail     3   end of dataBand3 </pre>	<p>Document tree</p> <pre> document1 ├── page1 │   ├── sideBySide1 │   │   ├── dataBand1 │   │   │   ├── detail1 │   │   │   └── detail1 │   │   ├── dataBand2 │   │   │   ├── detail2 │   │   │   └── detail2 │   │   └── dataBand3 │   │       ├── detail3 │   │       └── detail3 </pre>	
<pre> sideBySide1:SideBySide dataBand1:DataBand   detail1:Detail     1   end of dataBand1 bandContainer1:BandContainer   dataBand2:DataBand   detail2:Detail     2   end of dataBand2 dataBand3:DataBand   detail3:Detail     3   end of dataBand3 </pre>	<p>Document tree</p> <pre> document1 ├── page1 │   ├── sideBySide1 │   │   ├── dataBand1 │   │   │   ├── detail1 │   │   │   └── detail1 │   │   └── bandContainer1 │   │       ├── dataBand2 │   │       │   ├── detail2 │   │       │   └── detail2 │   │       └── dataBand3 │   │           ├── detail3 │   │           └── detail3 </pre>	



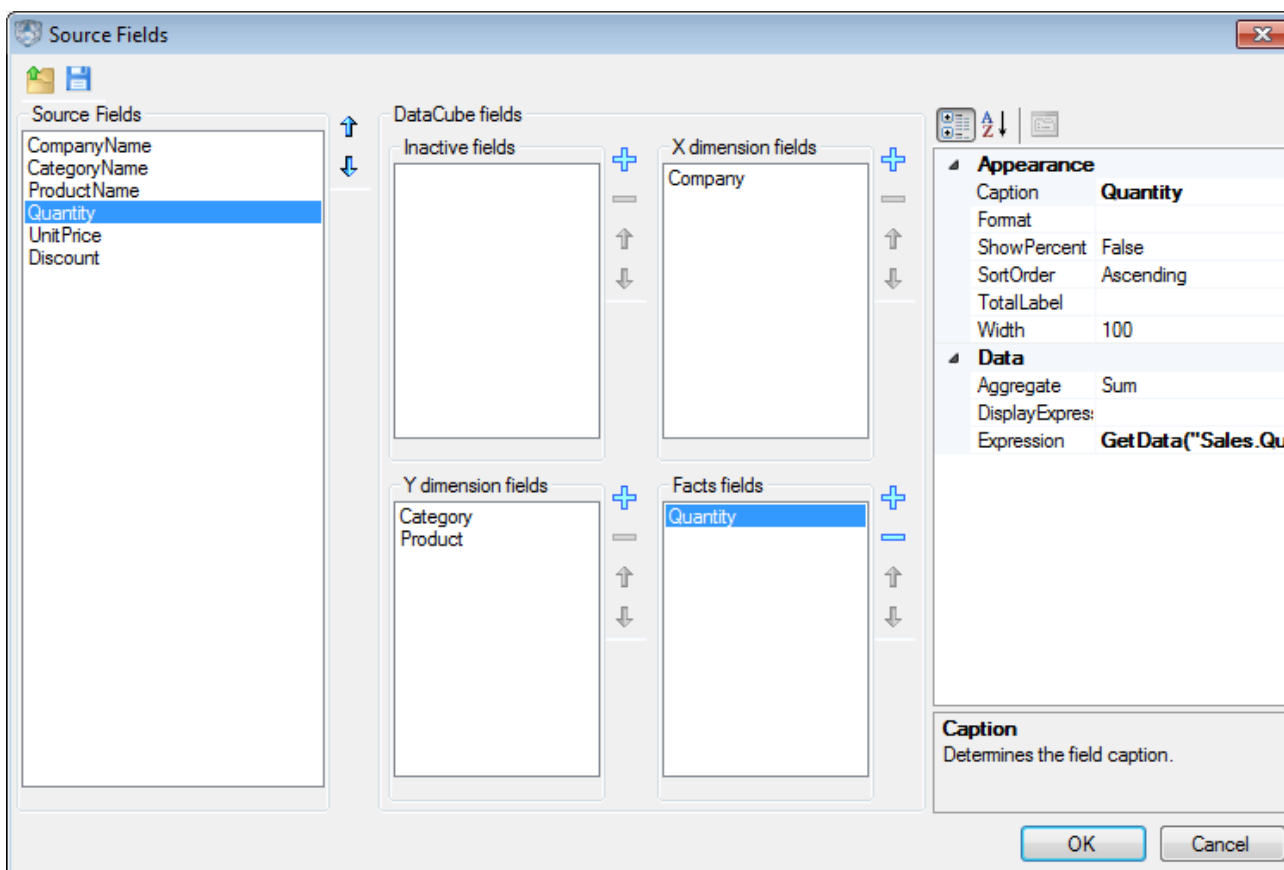


### Pivot Table

Pivot tables are used for structured data output and calculation of the total values. The PivotTable is used for the creation of pivot tables. The DataSource property sets the data source for the pivot table.

<b>Data</b>	
DataSource	Sales
Layout	
NestedFields	(Collection)
Tag	

The Layout property sets the rows and columns values of the result table. The property can be edited in Sources Fields.



The list of fields from data source is displayed on the left of the Source fields' area. The DataCube fields section is intended for data assignment in the pivot table. The fields of the table horizontal headline are set in X dimension fields list. The fields of the table vertical headline are set in Y dimension fields list. The Facts fields list includes fields for the cells with data. Property Grid is located to the left. It is used to set cells properties.

<b>Appearance</b>	
Caption	ProductName
Format	
ShowPercent	False
SortOrder	Ascending
TotalLabel	
Width	100
<b>Data</b>	
Aggregate	Sum
DisplayExpressi	
Expression	GetData("Sales.Pro



### Table Titles

The TableCaption property sets the table caption.

Property		Report				
StyleName	New Style			Company		
Styles						
TableCaption		Category	Product	Futterkiste	Karkki Oy	Total
ToolTip		Beverages	Chai	452.63		452.63
			Ipoh Coffee		909.00	909.00
			Stout	420.00		420.00
StyleName	New Style			Sales		
Styles				Company		
TableCaption	<b>Sales</b>	Category	Product	Futterkiste	Karkki Oy	Total
ToolTip		Beverages	Chai	452.625		452.625
			Ipoh Coffee		909	909
			Stout	420		420

The captions of the table rows and columns can be set in the Layout property editor of the PivotTable. The Caption property is used for this purpose. The TotalLabel sets the title for the column and row with total values. If this property is not set the total values are identified as Total.

Property		Report			
<b>Appearance</b>				CompanyName	
Caption	<b>CompanyName</b>				
Format		Product	Futterkiste	Karkki Oy	Total
ShowPercent	False	Chai	452.63		452.63
SortOrder	Ascending	Ipoh Coffee		909.00	909.00
TotalLabel		Stout	420.00		420.00
Width	100	Total	872.63	909.00	1781.63
<b>Appearance</b>				Company	
Caption	<b>Company</b>				
Format		Product	Futterkiste	Karkki Oy	Sum
ShowPercent	False	Chai	452.63		452.63
SortOrder	Ascending	Ipoh Coffee		909.00	909.00
TotalLabel	<b>Sum</b>	Stout	420.00		420.00
Width	100	Total	872.63	909.00	1781.63



The DimensionOnEveryPage property of the PivotTable sets the output of pivot table header on each page.

Property		Report																																																								
<b>Appearance</b> AutoColumnWidth: True AutoRowHeight: <b>None</b> DimensionOnEveryPage: False RowHeight: <b>70</b>		<table border="1"> <thead> <tr> <th colspan="2"></th> <th colspan="3">Sales</th> </tr> <tr> <th colspan="2"></th> <th colspan="2">Company</th> <th></th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Karkki Oy</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td rowspan="2">Beverages</td> <td>Chai</td> <td>452.63</td> <td></td> <td>452.63</td> </tr> <tr> <td>Ipoh Coffee</td> <td></td> <td>909.00</td> <td>909.00</td> </tr> <tr> <td colspan="5"><hr/></td> </tr> <tr> <td rowspan="2">Beverages</td> <td>Stout</td> <td>420.00</td> <td></td> <td>420.00</td> </tr> <tr> <td>Total</td> <td>872.63</td> <td>909.00</td> <td>1781.63</td> </tr> <tr> <td rowspan="2">Dairy Products</td> <td>Courdavault</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td>Total</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td colspan="2">Total</td> <td>1763.25</td> <td>909.00</td> <td>2672.25</td> </tr> </tbody> </table>						Sales					Company			Category	Product	Futterkiste	Karkki Oy	Total	Beverages	Chai	452.63		452.63	Ipoh Coffee		909.00	909.00	<hr/>					Beverages	Stout	420.00		420.00	Total	872.63	909.00	1781.63	Dairy Products	Courdavault	890.63		890.63	Total	890.63		890.63	Total		1763.25	909.00	2672.25	
		Sales																																																								
		Company																																																								
Category	Product	Futterkiste	Karkki Oy	Total																																																						
Beverages	Chai	452.63		452.63																																																						
	Ipoh Coffee		909.00	909.00																																																						
<hr/>																																																										
Beverages	Stout	420.00		420.00																																																						
	Total	872.63	909.00	1781.63																																																						
Dairy Products	Courdavault	890.63		890.63																																																						
	Total	890.63		890.63																																																						
Total		1763.25	909.00	2672.25																																																						
<b>Appearance</b> AutoColumnWidth: True AutoRowHeight: <b>None</b> DimensionOnEveryPage: <b>True</b> RowHeight: <b>70</b>		<table border="1"> <thead> <tr> <th colspan="2"></th> <th colspan="3">Sales</th> </tr> <tr> <th colspan="2"></th> <th colspan="2">Company</th> <th></th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Karkki Oy</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td rowspan="2">Beverages</td> <td>Chai</td> <td>452.63</td> <td></td> <td>452.63</td> </tr> <tr> <td>Ipoh Coffee</td> <td></td> <td>909.00</td> <td>909.00</td> </tr> <tr> <td colspan="5"><hr/></td> </tr> <tr> <th colspan="2"></th> <th colspan="3">Sales</th> </tr> <tr> <th colspan="2"></th> <th colspan="2">Company</th> <th></th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Karkki Oy</th> <th>Total</th> </tr> <tr> <td rowspan="2">Beverages</td> <td>Stout</td> <td>420.00</td> <td></td> <td>420.00</td> </tr> <tr> <td>Total</td> <td>872.63</td> <td>909.00</td> <td>1781.63</td> </tr> </tbody> </table>						Sales					Company			Category	Product	Futterkiste	Karkki Oy	Total	Beverages	Chai	452.63		452.63	Ipoh Coffee		909.00	909.00	<hr/>							Sales					Company			Category	Product	Futterkiste	Karkki Oy	Total	Beverages	Stout	420.00		420.00	Total	872.63	909.00	1781.63
		Sales																																																								
		Company																																																								
Category	Product	Futterkiste	Karkki Oy	Total																																																						
Beverages	Chai	452.63		452.63																																																						
	Ipoh Coffee		909.00	909.00																																																						
<hr/>																																																										
		Sales																																																								
		Company																																																								
Category	Product	Futterkiste	Karkki Oy	Total																																																						
Beverages	Stout	420.00		420.00																																																						
	Total	872.63	909.00	1781.63																																																						

It is possible to set the display of fact column title using the ShowSingleFact property.

Property		Report																										
Shift: <b>0 cm</b> ShowRepeatText: True ShowSingleFact: False ShowTableCaption: True		<table border="1"> <thead> <tr> <th colspan="2"></th> <th colspan="2">Sales</th> </tr> <tr> <th colspan="2"></th> <th colspan="2">Company</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td rowspan="2">Beverages</td> <td>Chai</td> <td>452.63</td> <td>452.63</td> </tr> <tr> <td>Stout</td> <td>420.00</td> <td>420.00</td> </tr> </tbody> </table>						Sales				Company		Category	Product	Futterkiste	Total	Beverages	Chai	452.63	452.63	Stout	420.00	420.00				
		Sales																										
		Company																										
Category	Product	Futterkiste	Total																									
Beverages	Chai	452.63	452.63																									
	Stout	420.00	420.00																									
Shift: <b>0 cm</b> ShowRepeatText: True ShowSingleFact: <b>True</b> ShowTableCaption: True		<table border="1"> <thead> <tr> <th colspan="2"></th> <th colspan="2">Sales</th> </tr> <tr> <th colspan="2"></th> <th colspan="2">Company</th> </tr> <tr> <th colspan="2"></th> <th>Futterkiste</th> <th>Total</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Price</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td rowspan="2">Beverages</td> <td>Chai</td> <td>452.63</td> <td>452.63</td> </tr> <tr> <td>Stout</td> <td>420.00</td> <td>420.00</td> </tr> </tbody> </table>						Sales				Company				Futterkiste	Total	Category	Product	Price	Price	Beverages	Chai	452.63	452.63	Stout	420.00	420.00
		Sales																										
		Company																										
		Futterkiste	Total																									
Category	Product	Price	Price																									
Beverages	Chai	452.63	452.63																									
	Stout	420.00	420.00																									



It is possible to set the repetition of the text on the next page if the string doesn't fit the current page. To do this you should use the ShowRepeatText property.

Property		Report		
RowHeight	30	Dairy Products	Courdavault	890.63
Shift	0 cm		Flotemysost	2256.00
ShowRepeatText	True		Geitost	154.84
ShowSingleFact	False			
		Dairy Products	Total	3301.47
RowHeight	30	Dairy Products	Courdavault	890.63
Shift	0 cm		Flotemysost	2256.00
ShowRepeatText	False		Geitost	154.84
ShowSingleFact	False			
			Total	3301.47

### Data Format

It is possible to define the format of the output data using the Format property of the Source Field editor.

Property		Report		
<b>Appearance</b>		452.625		452.625
Caption	Price		909	909
Format		420		420
ShowPercent	False	872.625	909	1781.625
SortOrder	Ascending	890.625		890.625
		890.625		890.625
		1763.25	909	2672.25
<b>Appearance</b>		452.63		452.63
Caption	Price		909.00	909.00
Format	0.00	420.00		420.00
ShowPercent	False	872.63	909.00	1781.63
SortOrder	Ascending	890.63		890.63
		890.63		890.63
		1763.25	909.00	2672.25
<b>Appearance</b>		16.94%		16.94%
Caption	Price		34.02%	34.02%
Format	0.00	15.72%		15.72%
ShowPercent	True	32.66%	34.02%	66.67%
SortOrder	Ascending	33.33%		33.33%
		33.33%		33.33%
		65.98%	34.02%	100.00%



The *ShowZeroValues* property is responsible for display of NULL values in a report.

Property		Report		
ShowSingleFact	False	452.63		452.63
ShowTableCaption	True		909.00	909.00
ShowZeroValues	False	420.00		420.00
StyleName	Main			
ShowSingleFact	False	452.63	0	452.63
ShowTableCaption	True	0	909.00	909.00
ShowZeroValues	True	420.00	0	420.00
StyleName	Main			

### Ability to Hide/Show Total

It is possible to hide the "Total" columns or rows of a pivot table using the *ShowTotal* property of the fields. The Source Field editor allows the setting of this property.

Property		Report		
<b>Appearance</b> Caption <b>Category</b> Format ShowPercent False ShowTotal True SortOrder Ascending		Category	Product	Alfreds Futterkiste
		Beverages	Chai	505,88
			Ipoh Coffee	
			Steeleye Stout	525,00
			Total	1030,88
		Confections	Maxilaku	130,00
			Total	130,00
		Total	4509,22	
<b>Appearance</b> Caption <b>Category</b> Format ShowPercent False ShowTotal False SortOrder Ascending		Category	Product	Alfreds Futterkiste
		Beverages	Chai	505,88
			Ipoh Coffee	
			Steeleye Stout	525,00
			Total	
		Confections	Maxilaku	130,00
			Total	4509,22

### Data Sorting

The order of sorting is set in the *SortOrder* property. Ascending (direct order) and Descending (reversed order) are available.

### Data Filtering

Filtering is used when you need to output data which address some specific conditions. The *FilterExpression* property is used to set the conditions. The expression is set by the script. The result of calculation of this expression will define data for the table.

Property	Report					
Not set			Futterkiste	Karkki Oy	Shopy	Total
	Category	Product	Discount	Discount	Discount	Discount
	Beverages	Chai	0.15	0.04		0.19
		Ipoh Coffee		0.1		0.1
		Stout	0.2			0.2
		Total	0.35	0.14		0.49
FilterExpression="(double)GetData("Sales.Discount") >= 0.1"			Futterkiste	Karkki Oy	Total	
	Category	Product	Discount	Discount	Discount	
	Beverages	Chai	0.15		0.15	
		Ipoh Coffee		0.1	0.1	
		Stout	0.2		0.2	
		Total	0.35	0.1	0.45	



### Table Size

The table cell size is defined automatically according to the table data size by default. The AutoColumnWidth (setting of column width) and AutoRowHeight (setting of rows height) are used for this. In order to set arbitrary row height you should select "None" value in the AutoRowHeight property and indicate the height value in pixels in the RowHeight property.

Appearance	
AutoColumnWidth	True
AutoRowHeight	<b>None</b>
DimensionOnEveryPage	False
RowHeight	<b>100</b>

Left margins are set by the Shift property.



### Table Styles

The StyleName property sets the table style. The Style Sheet Editor is used for styles creation.

It is possible to set separate styles for each cell type using the Style properties group.

Property		Report																																																		
StyleName <input type="checkbox"/> Styles EvenRowFactStyle FieldCaptionStyle GroupTotalStyle MainTotalStyle OddRowFactStyle TableCaptionStyle TotalStyle XDimensionStyle YDimensionStyle		<table border="1"> <thead> <tr> <th colspan="5">Sales</th> </tr> <tr> <th colspan="5">Company</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Karkki Oy</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td></td> <td>Chai</td> <td>452.63</td> <td></td> <td>452.63</td> </tr> <tr> <td>Beverages</td> <td>Ipoh Coffee</td> <td></td> <td>909.00</td> <td>909.00</td> </tr> <tr> <td></td> <td>Stout</td> <td>420.00</td> <td></td> <td>420.00</td> </tr> <tr> <td></td> <td>Total</td> <td>872.63</td> <td>909.00</td> <td>1781.63</td> </tr> <tr> <td>Dairy Products</td> <td>Courdavault</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td></td> <td>Total</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td>Total</td> <td></td> <td>1763.25</td> <td>909.00</td> <td>2672.25</td> </tr> </tbody> </table>	Sales					Company					Category	Product	Futterkiste	Karkki Oy	Total		Chai	452.63		452.63	Beverages	Ipoh Coffee		909.00	909.00		Stout	420.00		420.00		Total	872.63	909.00	1781.63	Dairy Products	Courdavault	890.63		890.63		Total	890.63		890.63	Total		1763.25	909.00	2672.25
Sales																																																				
Company																																																				
Category	Product	Futterkiste	Karkki Oy	Total																																																
	Chai	452.63		452.63																																																
Beverages	Ipoh Coffee		909.00	909.00																																																
	Stout	420.00		420.00																																																
	Total	872.63	909.00	1781.63																																																
Dairy Products	Courdavault	890.63		890.63																																																
	Total	890.63		890.63																																																
Total		1763.25	909.00	2672.25																																																
StyleName <b>Main</b> <input type="checkbox"/> Styles EvenRowFactStyle FieldCaptionStyle GroupTotalStyle MainTotalStyle OddRowFactStyle TableCaptionStyle TotalStyle XDimensionStyle YDimensionStyle		<table border="1"> <thead> <tr> <th colspan="5">Sales</th> </tr> <tr> <th colspan="5">Company</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Karkki Oy</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td></td> <td>Chai</td> <td>452.63</td> <td></td> <td>452.63</td> </tr> <tr> <td>Beverages</td> <td>Ipoh Coffee</td> <td></td> <td>909.00</td> <td>909.00</td> </tr> <tr> <td></td> <td>Stout</td> <td>420.00</td> <td></td> <td>420.00</td> </tr> <tr> <td></td> <td>Total</td> <td>872.63</td> <td>909.00</td> <td>1781.63</td> </tr> <tr> <td>Dairy Products</td> <td>Courdavault</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td></td> <td>Total</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td>Total</td> <td></td> <td>1763.25</td> <td>909.00</td> <td>2672.25</td> </tr> </tbody> </table>	Sales					Company					Category	Product	Futterkiste	Karkki Oy	Total		Chai	452.63		452.63	Beverages	Ipoh Coffee		909.00	909.00		Stout	420.00		420.00		Total	872.63	909.00	1781.63	Dairy Products	Courdavault	890.63		890.63		Total	890.63		890.63	Total		1763.25	909.00	2672.25
Sales																																																				
Company																																																				
Category	Product	Futterkiste	Karkki Oy	Total																																																
	Chai	452.63		452.63																																																
Beverages	Ipoh Coffee		909.00	909.00																																																
	Stout	420.00		420.00																																																
	Total	872.63	909.00	1781.63																																																
Dairy Products	Courdavault	890.63		890.63																																																
	Total	890.63		890.63																																																
Total		1763.25	909.00	2672.25																																																
StyleName <b>Main</b> <input type="checkbox"/> Styles EvenRowFactStyle <b>DataStyle</b> FieldCaptionStyle <b>CaptionStyle</b> GroupTotalStyle MainTotalStyle <b>Total Style</b> OddRowFactStyle <b>DataStyle</b> TableCaptionStyle <b>CaptionStyle</b> TotalStyle <b>Total Style</b> XDimensionStyle YDimensionStyle		<table border="1"> <thead> <tr> <th colspan="5">Sales</th> </tr> <tr> <th colspan="5">Company</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Futterkiste</th> <th>Karkki Oy</th> <th>Total</th> </tr> </thead> <tbody> <tr> <td></td> <td>Chai</td> <td>452.63</td> <td></td> <td>452.63</td> </tr> <tr> <td>Beverages</td> <td>Ipoh Coffee</td> <td></td> <td>909.00</td> <td>909.00</td> </tr> <tr> <td></td> <td>Stout</td> <td>420.00</td> <td></td> <td>420.00</td> </tr> <tr> <td></td> <td>Total</td> <td>872.63</td> <td>909.00</td> <td>1781.63</td> </tr> <tr> <td>Dairy Products</td> <td>Courdavault</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td></td> <td>Total</td> <td>890.63</td> <td></td> <td>890.63</td> </tr> <tr> <td>Total</td> <td></td> <td>1763.25</td> <td>909.00</td> <td>2672.25</td> </tr> </tbody> </table>	Sales					Company					Category	Product	Futterkiste	Karkki Oy	Total		Chai	452.63		452.63	Beverages	Ipoh Coffee		909.00	909.00		Stout	420.00		420.00		Total	872.63	909.00	1781.63	Dairy Products	Courdavault	890.63		890.63		Total	890.63		890.63	Total		1763.25	909.00	2672.25
Sales																																																				
Company																																																				
Category	Product	Futterkiste	Karkki Oy	Total																																																
	Chai	452.63		452.63																																																
Beverages	Ipoh Coffee		909.00	909.00																																																
	Stout	420.00		420.00																																																
	Total	872.63	909.00	1781.63																																																
Dairy Products	Courdavault	890.63		890.63																																																
	Total	890.63		890.63																																																
Total		1763.25	909.00	2672.25																																																

### Pivot Table Saving

The Layout property sets expressions and position of fields in pivot table. It can be saved to the file for further use and quick return to the initial pivot table form. You can use the "Save" and "Open" buttons of the Layout property editor for saving and loading of the table parameters. The file has the XML format and .dcl extension.



## Image

The Picture element is used for images display.

The Image property sets the image. In order to select file you should press the button near the Image property or double click with left mouse button on the area of Picture element. It is possible to upload the image of the following formats: \*.bmp, \*.gif, \*.jpg, \*.jpeg, \*.png, \*.ico, \*.emf, \*.wmf.

### Size and Scale of the Image

The Size property sets the size of the Picture element. If the image size doesn't fit Picture element size during the output, you can use the SizeMode property to change sizes.

The image extension – the image changes the proportions depending on the elements sizes:

Property		Report								
<table border="1"> <tr> <td>⊕ Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>SizeMode</td> <td>Stretch</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	⊕ Margins	0; 0; 0; 0 cm	SizeMode	Stretch	StyleName		ToolTip			
⊕ Margins	0; 0; 0; 0 cm									
SizeMode	Stretch									
StyleName										
ToolTip										
The image cutting – the image saves its natural size and is cut according to the element's size:										
<table border="1"> <tr> <td>⊕ Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>SizeMode</td> <td>Normal</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	⊕ Margins	0; 0; 0; 0 cm	SizeMode	Normal	StyleName		ToolTip			
⊕ Margins	0; 0; 0; 0 cm									
SizeMode	Normal									
StyleName										
ToolTip										
Auto size – the element's size is changed according to the image size and is not edited:										
<table border="1"> <tr> <td>⊕ Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>SizeMode</td> <td>AutoSize</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	⊕ Margins	0; 0; 0; 0 cm	SizeMode	AutoSize	StyleName		ToolTip			
⊕ Margins	0; 0; 0; 0 cm									
SizeMode	AutoSize									
StyleName										
ToolTip										
Uniform – the image is resized to maximum size inside element bounds, image's aspect ratio is preserved:										
<table border="1"> <tr> <td>⊕ Margins</td> <td>0; 0; 0; 0 cm</td> </tr> <tr> <td>SizeMode</td> <td>Uniform</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> <tr> <td>ToolTip</td> <td></td> </tr> </table>	⊕ Margins	0; 0; 0; 0 cm	SizeMode	Uniform	StyleName		ToolTip			
⊕ Margins	0; 0; 0; 0 cm									
SizeMode	Uniform									
StyleName										
ToolTip										





### Image Alignment

If "Normal" or "Uniform" size mode is chosen, then the ImageAlign property will define the image position within bounds of the element.



Property	Report										
<table border="1"><tr><td>Fill</td><td>[None]</td></tr><tr><td>+ Image</td><td> System.Drawing.Bitmap</td></tr><tr><td>ImageAlign</td><td>TopLeft</td></tr><tr><td>+ Margins</td><td>0; 0; 0; 0 cm</td></tr><tr><td>SizeMode</td><td>Normal</td></tr></table>	Fill	[None]	+ Image	System.Drawing.Bitmap	ImageAlign	TopLeft	+ Margins	0; 0; 0; 0 cm	SizeMode	Normal	
Fill	[None]										
+ Image	System.Drawing.Bitmap										
ImageAlign	TopLeft										
+ Margins	0; 0; 0; 0 cm										
SizeMode	Normal										
<table border="1"><tr><td>+ Image</td><td> System.Drawing.Bitmap</td></tr><tr><td>ImageAlign</td><td>MiddleCenter</td></tr><tr><td>+ Margins</td><td>0; 0; 0; 0 cm</td></tr><tr><td>SizeMode</td><td>Normal</td></tr><tr><td>StyleName</td><td></td></tr></table>	+ Image	System.Drawing.Bitmap	ImageAlign	MiddleCenter	+ Margins	0; 0; 0; 0 cm	SizeMode	Normal	StyleName		
+ Image	System.Drawing.Bitmap										
ImageAlign	MiddleCenter										
+ Margins	0; 0; 0; 0 cm										
SizeMode	Normal										
StyleName											
<table border="1"><tr><td>+ Image</td><td> System.Drawing.Bitmap</td></tr><tr><td>ImageAlign</td><td>BottomRight</td></tr><tr><td>+ Margins</td><td>0; 0; 0; 0 cm</td></tr><tr><td>SizeMode</td><td>Normal</td></tr><tr><td>StyleName</td><td></td></tr></table>	+ Image	System.Drawing.Bitmap	ImageAlign	BottomRight	+ Margins	0; 0; 0; 0 cm	SizeMode	Normal	StyleName		
+ Image	System.Drawing.Bitmap										
ImageAlign	BottomRight										
+ Margins	0; 0; 0; 0 cm										
SizeMode	Normal										
StyleName											



### Dynamic Image Loading

When it is needed to load the image in the process of report generation, you can open the Bindings properties and use the Image property. Use the following script when it is needed to load image from file:



```
System.Drawing.Image.FromFile("<path to the file>")
```

Report Template	Report
<pre>dataBand:DataBand DataSource = Employees; detail:Detail &lt;dataBand["EmployeeName"]&gt; end of dataBand</pre> <pre>dataBand.DataSource = Employees; textBox.Value = dataBand["EmployeeName"]; picture.Image = System.Drawing.Image.FromFile ("C:\\Data\\Pictures\\"+dataBand["EmployeeName"]. ToString()+".png"); picture.SizeMode = AutoSize;</pre>	<p><u>Nancy Davolio</u></p>  <p><u>Andrew Fuller</u></p> 



### Image from Data Source

To output images from data source you should put the Picture element to Detail which is located in the DataBand with the set data source, and then set the Image property on the Bindings tab. In this case the images are stored in the data source in byte representation.

Report Template	Report
<pre>dataBand:DataBand DataSource = Employee detail:Detail &lt;dataBand["EmployeeName"]&gt; end of dataBand</pre> <pre>dataBand.DataSource = Employees; textBox.Value = dataBand["EmployeeName"]; picture.Image = dataBand["Picture"]; picture.SizeMode = AutoSize;</pre>	<p><u>Nancy Davolio</u></p>  <p><u>Andrew Fuller</u></p> 

## Using Shapes

Shape element can be used to output various graphical figures.

### Shape Types

ShapeStyle property defines the type of a figure. The following types are available:

Shape Styles	Report																		
<p>RectangleShape – figure draws a rectangle.</p> <table border="1"> <tr> <td colspan="2"><b>Appearance</b></td> </tr> <tr> <td>Fill</td> <td>HatchFill</td> </tr> <tr> <td>Line</td> <td>2 Solid DodgerBlue</td> </tr> <tr> <td>Shadow</td> <td>Shadow</td> </tr> <tr> <td>ShapeStyle</td> <td>RectangleShape</td> </tr> </table>	<b>Appearance</b>		Fill	HatchFill	Line	2 Solid DodgerBlue	Shadow	Shadow	ShapeStyle	RectangleShape									
<b>Appearance</b>																			
Fill	HatchFill																		
Line	2 Solid DodgerBlue																		
Shadow	Shadow																		
ShapeStyle	RectangleShape																		
<p>TriangleShape – figure draws an isosceles triangle. Direction property sets triangle orientation.</p> <table border="1"> <tr> <td colspan="2"><b>Appearance</b></td> </tr> <tr> <td>Fill</td> <td>HatchFill</td> </tr> <tr> <td>Line</td> <td>2 Solid DeepPink</td> </tr> <tr> <td>Shadow</td> <td>Shadow</td> </tr> <tr> <td>ShapeStyle</td> <td>TriangleShape</td> </tr> <tr> <td>Direction</td> <td>Right</td> </tr> <tr> <td>StyleName</td> <td></td> </tr> </table>	<b>Appearance</b>		Fill	HatchFill	Line	2 Solid DeepPink	Shadow	Shadow	ShapeStyle	TriangleShape	Direction	Right	StyleName						
<b>Appearance</b>																			
Fill	HatchFill																		
Line	2 Solid DeepPink																		
Shadow	Shadow																		
ShapeStyle	TriangleShape																		
Direction	Right																		
StyleName																			
<p>CrossShape – figure draws a cross.</p> <table border="1"> <tr> <td colspan="2"><b>Appearance</b></td> </tr> <tr> <td>Fill</td> <td>HatchFill</td> </tr> <tr> <td>Line</td> <td>2 Solid DarkTurquoise</td> </tr> <tr> <td>Shadow</td> <td>Shadow</td> </tr> <tr> <td>ShapeStyle</td> <td>CrossShape</td> </tr> </table>	<b>Appearance</b>		Fill	HatchFill	Line	2 Solid DarkTurquoise	Shadow	Shadow	ShapeStyle	CrossShape									
<b>Appearance</b>																			
Fill	HatchFill																		
Line	2 Solid DarkTurquoise																		
Shadow	Shadow																		
ShapeStyle	CrossShape																		
<p>DiamondShape – figure draws a diamond.</p> <table border="1"> <tr> <td colspan="2"><b>Appearance</b></td> </tr> <tr> <td>Fill</td> <td>HatchFill</td> </tr> <tr> <td>Line</td> <td>2 Solid DarkViolet</td> </tr> <tr> <td>Shadow</td> <td>Shadow</td> </tr> <tr> <td>ShapeStyle</td> <td>DiamondShape</td> </tr> </table>	<b>Appearance</b>		Fill	HatchFill	Line	2 Solid DarkViolet	Shadow	Shadow	ShapeStyle	DiamondShape									
<b>Appearance</b>																			
Fill	HatchFill																		
Line	2 Solid DarkViolet																		
Shadow	Shadow																		
ShapeStyle	DiamondShape																		
<p>StarShape – figure draws a star. InternalRadius property sets a radius of an inscribed circle, KeepRoundForm property defines star shape – ellipse or a circle, Points property sets number of the rays, StartAngle property sets the rotation angle of a star.</p> <table border="1"> <tr> <td colspan="2"><b>Appearance</b></td> </tr> <tr> <td>Fill</td> <td>HatchFill</td> </tr> <tr> <td>Line</td> <td>2 Solid Salmon</td> </tr> <tr> <td>Shadow</td> <td>Shadow</td> </tr> <tr> <td>ShapeStyle</td> <td>StarShape</td> </tr> <tr> <td>InternalRadius</td> <td>0,37999999523162842</td> </tr> <tr> <td>KeepRoundForm</td> <td>False</td> </tr> <tr> <td>Points</td> <td>5</td> </tr> <tr> <td>StartAngle</td> <td>0</td> </tr> </table>	<b>Appearance</b>		Fill	HatchFill	Line	2 Solid Salmon	Shadow	Shadow	ShapeStyle	StarShape	InternalRadius	0,37999999523162842	KeepRoundForm	False	Points	5	StartAngle	0	
<b>Appearance</b>																			
Fill	HatchFill																		
Line	2 Solid Salmon																		
Shadow	Shadow																		
ShapeStyle	StarShape																		
InternalRadius	0,37999999523162842																		
KeepRoundForm	False																		
Points	5																		
StartAngle	0																		
<p>ArrowShape – figure draws an arrow. Direction property sets the direction of an arrow.</p> <table border="1"> <tr> <td colspan="2"><b>Appearance</b></td> </tr> <tr> <td>Fill</td> <td>HatchFill</td> </tr> <tr> <td>Line</td> <td>2 Solid SpringGreen</td> </tr> <tr> <td>Shadow</td> <td>Shadow</td> </tr> <tr> <td>ShapeStyle</td> <td>ArrowShape</td> </tr> <tr> <td>Direction</td> <td>Left</td> </tr> </table>	<b>Appearance</b>		Fill	HatchFill	Line	2 Solid SpringGreen	Shadow	Shadow	ShapeStyle	ArrowShape	Direction	Left							
<b>Appearance</b>																			
Fill	HatchFill																		
Line	2 Solid SpringGreen																		
Shadow	Shadow																		
ShapeStyle	ArrowShape																		
Direction	Left																		
<p>RectTriangleShape – figure draws a pentagon. Direction property sets the direction of a figure.</p>																			

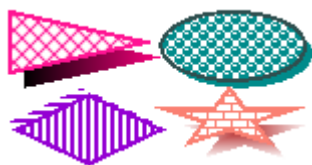


<table border="1"> <tr><td>[-] Appearance</td></tr> <tr><td>+ Fill</td></tr> <tr><td>+ Line</td></tr> <tr><td>+ Shadow</td></tr> <tr><td>[-] ShapeStyle</td></tr> <tr><td>  Direction</td></tr> </table>	[-] Appearance	+ Fill	+ Line	+ Shadow	[-] ShapeStyle	Direction	<table border="1"> <tr><td>HatchFill</td></tr> <tr><td>2 Solid DarkOrange</td></tr> <tr><td>Shadow</td></tr> <tr><td>RectTriangleShape</td></tr> <tr><td>Up</td></tr> </table>	HatchFill	2 Solid DarkOrange	Shadow	RectTriangleShape	Up	
[-] Appearance													
+ Fill													
+ Line													
+ Shadow													
[-] ShapeStyle													
Direction													
HatchFill													
2 Solid DarkOrange													
Shadow													
RectTriangleShape													
Up													
EllipseShape – figure draws ellipse.													
<table border="1"> <tr><td>[-] Appearance</td></tr> <tr><td>+ Fill</td></tr> <tr><td>+ Line</td></tr> <tr><td>+ Shadow</td></tr> <tr><td>[-] ShapeStyle</td></tr> </table>	[-] Appearance	+ Fill	+ Line	+ Shadow	[-] ShapeStyle	<table border="1"> <tr><td>HatchFill</td></tr> <tr><td>2 Solid DarkCyan</td></tr> <tr><td>Shadow</td></tr> <tr><td>EllipseShape</td></tr> </table>	HatchFill	2 Solid DarkCyan	Shadow	EllipseShape			
[-] Appearance													
+ Fill													
+ Line													
+ Shadow													
[-] ShapeStyle													
HatchFill													
2 Solid DarkCyan													
Shadow													
EllipseShape													
RoundRectangleShape – figure draws a rectangle with truncated corners. Round property sets truncation radius.													
<table border="1"> <tr><td>[-] Appearance</td></tr> <tr><td>+ Fill</td></tr> <tr><td>+ Line</td></tr> <tr><td>+ Shadow</td></tr> <tr><td>[-] ShapeStyle</td></tr> <tr><td>  Round</td></tr> </table>	[-] Appearance	+ Fill	+ Line	+ Shadow	[-] ShapeStyle	Round	<table border="1"> <tr><td>HatchFill</td></tr> <tr><td>2 Solid DarkSlateBlue</td></tr> <tr><td>Shadow</td></tr> <tr><td>RoundRectangleShape</td></tr> <tr><td>0.4</td></tr> </table>	HatchFill	2 Solid DarkSlateBlue	Shadow	RoundRectangleShape	0.4	
[-] Appearance													
+ Fill													
+ Line													
+ Shadow													
[-] ShapeStyle													
Round													
HatchFill													
2 Solid DarkSlateBlue													
Shadow													
RoundRectangleShape													
0.4													
ParallelogramShape – figure draws a parallelogram. Angle property sets a rotation angle of a parallelogram.													
<table border="1"> <tr><td>[-] Appearance</td></tr> <tr><td>+ Fill</td></tr> <tr><td>+ Line</td></tr> <tr><td>+ Shadow</td></tr> <tr><td>[-] ShapeStyle</td></tr> <tr><td>  Angle</td></tr> </table>	[-] Appearance	+ Fill	+ Line	+ Shadow	[-] ShapeStyle	Angle	<table border="1"> <tr><td>HatchFill</td></tr> <tr><td>2 Solid DarkMagenta</td></tr> <tr><td>Shadow</td></tr> <tr><td>ParallelogramShape</td></tr> <tr><td>45</td></tr> </table>	HatchFill	2 Solid DarkMagenta	Shadow	ParallelogramShape	45	
[-] Appearance													
+ Fill													
+ Line													
+ Shadow													
[-] ShapeStyle													
Angle													
HatchFill													
2 Solid DarkMagenta													
Shadow													
ParallelogramShape													
45													
LineShape – figure draws a line. LineKind property sets a rotation angle of a line.													
<table border="1"> <tr><td>[-] Appearance</td></tr> <tr><td>  Fill</td></tr> <tr><td>+ Line</td></tr> <tr><td>+ Shadow</td></tr> <tr><td>[-] ShapeStyle</td></tr> <tr><td>  LineKind</td></tr> </table>	[-] Appearance	Fill	+ Line	+ Shadow	[-] ShapeStyle	LineKind	<table border="1"> <tr><td>[None]</td></tr> <tr><td>2 Solid Red</td></tr> <tr><td>Shadow</td></tr> <tr><td>LineShape</td></tr> <tr><td>BackSlash</td></tr> </table>	[None]	2 Solid Red	Shadow	LineShape	BackSlash	
[-] Appearance													
Fill													
+ Line													
+ Shadow													
[-] ShapeStyle													
LineKind													
[None]													
2 Solid Red													
Shadow													
LineShape													
BackSlash													

### Shape Shadow

Figure shadows sets by a Shadow property. DX and DY values set the shadow offset, Fill property sets the fill. Adding a shadow leaves the size of an element the same while the size of a figure decreases.

Example of a figure shadow usage:



### Shape Border

Figure border is defined by the Line property. Color property sets the color of a line, Style property sets line style, and Width property sets line width.



Examples of figure lines:





## Barcodes

BarCode element is used to create barcodes. Code for barcode is set in the Code property. Digits can be used for the most types of barcodes; Latin symbols can be used for some advanced barcode types.

Property		Report
CalcCheckSumm	False	 123456
Code	123456	
CodeType	Code39	
CalcCheckSumm	False	 Code 123456
Code	Code 123456	
CodeType	Code39Extended	

Barcode ratio is set in the BarRatio property.

## Barcode Types

Type of the code is set in the CodeType property. The following linear barcode types are available: Code\_2\_5\_interleaved, Code\_2\_5\_industrial, Code\_2\_5\_matrix, Code39 (default value), Code39Extended, Code128A, Code128B, Code128C, Code93, Code93Extended, CodeMSI, CodePostNet, CodeCodabar, CodeEAN8, CodeJAN8, CodeEAN13, CodeJAN13, CodeUPC\_A, CodeUPC\_E0, CodeUPC\_E1, CodeUPC\_Supp2, CodeUPC\_Supp5, CodeEAN128A, CodeEAN128B, CodeEAN128C.

2D barcodes are use with additional settings:

PDF417, PDF417Compact. Settings for these barcode types are in the Pdf417Settings properties group:

CodeType	<b>PDF417</b>
Fill	[None]
Font	Arial; 12pt
LabelPosition	BottomCenter
Margins	0; 0; 0; 0 cm
MicroQRCodeSettings	PerpetuumSoft.Framework.Drawir
Pdf417Settings	3; 0; 5; TextMode
ColumnsCount	3
ErrorLevel	0
RowsCount	5
Type	TextMode

QRCode. QRCodeSettings property group:

CodeType	<b>QRCode</b>
Fill	[None]
Font	Arial; 12pt
LabelPosition	BottomCenter
Margins	0; 0; 0; 0 cm
MicroQRCodeSettings	PerpetuumSoft.Framework.Drawir
Pdf417Settings	3; 0; 5; TextMode
QRCodeSettings	PerpetuumSoft.Framework.Drawi
ErrorCorrectionLevel	L
Mask	Mask000
MirrorImaging	False
ReflectanceReversal	False
Version	4


MicroQRCode. MicroQRCodeSettings property group:



CodeType	<b>MicroQRCode</b>
Fill	[None]
Font	Arial; 12pt
LabelPosition	BottomCenter
Margins	0; 0; 0; 0 cm
MicroQRCodeSettings	PerpetuumSoft.Framework.Drawi
ErrorCorrectionLevel	L
Mask	Mask00
MirrorImaging	False
ReflectanceReversal	False
Version	4



Japan\_Postal barcode type should be mentioned separately. Amount of points for the Japan\_Postal barcode is set in the BarPoint property. Size of the barcode is standard and it depends on the BarPoint property.

Property	Report
BarPoint	<b>10</b>
BarRatio	2
Border	
CalcCheckSumm	False
Code	<b>123456</b>
CodeType	<b>Japan_Postal</b>





### Barcode Style

To create background color of the barcode BackColor property is used. It is possible to set color of the barcode itself in the BarColor property.

Property	Report
BackColor	<input type="checkbox"/> Transparent
BarColor	<input checked="" type="checkbox"/> Black
	 123456
BackColor	<input checked="" type="checkbox"/> <b>DodgerBlue</b>
BarColor	<input checked="" type="checkbox"/> <b>DeepPink</b>
	 123456

ShowLabel property defines displaying of the Label in the barcode. It is possible to set font (Font property), color (TextColor property) and position of the label (LabelPosition property).

Property	Report
Font	Calibri; 9pt; Bold=Off; Italic=Off; U
LabelPosition	<b>TopLeft</b>
Margins	0; 0; 0; 0 cm
	 123456
ShowCheckDigit	False
ShowLabel	<b>False</b>
StyleName	
	

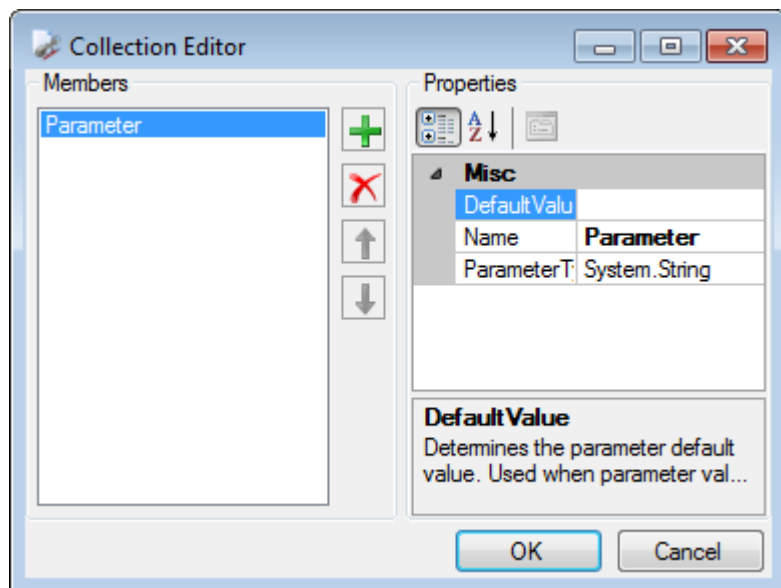
### CalcCheckSumm and ShowCheckDigit Properties

TDB



## Parameters

Parameters property is used to set report parameters. Collection Editor is an editor of this property.



DefaultValue property sets parameter value. Name property sets name to refer to parameter value in subreport. ParameterType property sets type of the parameter (string type is used by default for all parameters).

To get parameter value in subreport the following expression is used:  
GetParameter("<ParameterName>")

TBD





### Hyperlinks and Report Navigation

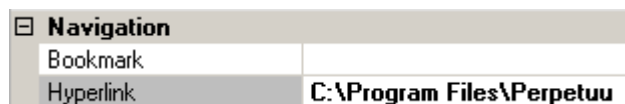
Hyperlink property of the objects is used to set hyperlinks.

#### Calling External Applications

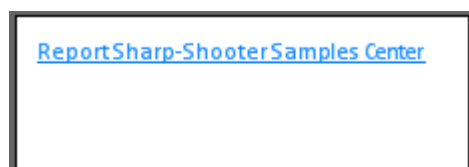
Hyperlink property can be used to call external applications.

Here is the sample of using TextBox element property to call application:

Property `textBox.Hyperlink` = "C:\Program Files\Perpetuum Software\Net ModelKit Suite\Bin\ReportSharpShooterSamplesCenter.exe".



Report:



Now when you click TextBox area, SharpShooter Reports Samples Center will run.

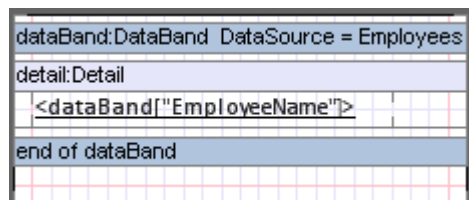
#### Opening Files

Files to which links are set in the Hyperlink property are opened with the help of programs associated with the file type.

Sample of setting links to files:

Hyperlink property is set by the script that forms file address in the process of report generation.

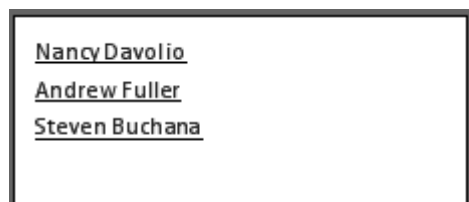
Report Template:



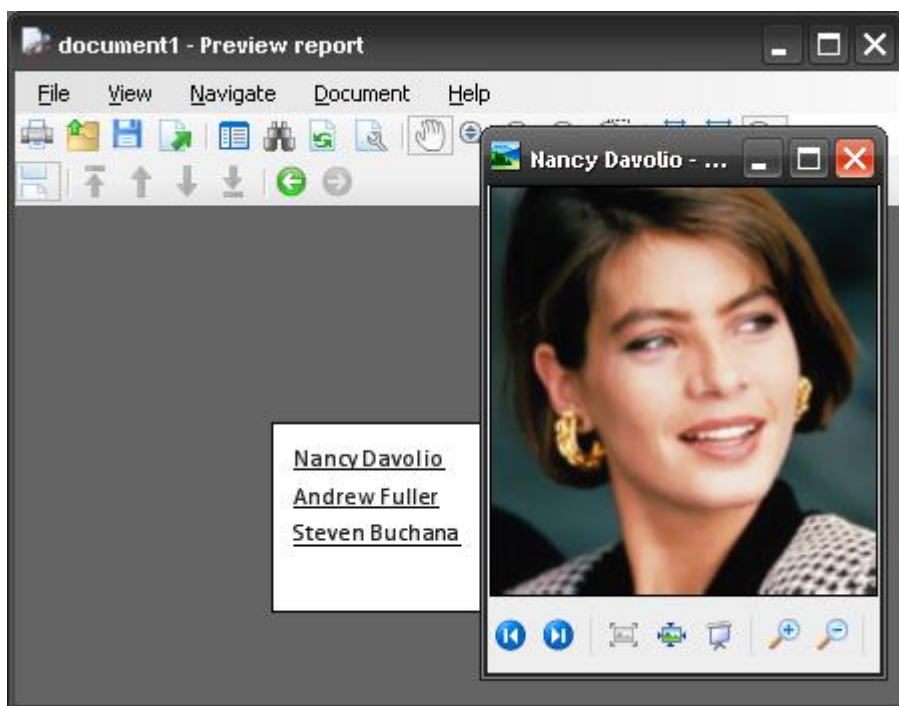
`dataBand.DataSource = Employees;`

`textBox.Value = dataBand["EmployeeName"]; textBox.Hyperlink = "C:\\Data\\Pictures\\"+dataBand["EmployeeName"].ToString()+".png"`

Report:



When you click on the TextBox element area the corresponding picture is opened with the help of picture viewer:



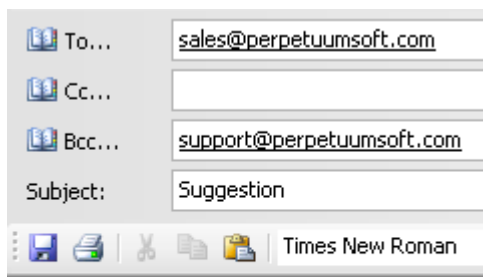
### Hyperlinks to Internet Resources and Emails

To get hyperlinks to Internet resources in the Hyperlink property it is necessary to set URL. For example, `textBox.Hyperlink = "http://www.perpetuumsoft.com/"`

Navigation	
Bookmark	
Hyperlink	<a href="http://www.perpetuumsoft.com/">http://www.perpetuumsoft.com/</a>

When you click on the TextBox element area in the final document, hyperlink is opened with the help of browser.

To make email client open when you click the link, Hyperlink property should contain `mailto` hyperlink, for example, `textBox.Hyperlink = "mailto:sales@perpetuumsoft.com"`. Then click on the element will open email client to send message to the set address. The same with the parameters (subject, body, cc, bcc,); it is possible to set subject, content of the message, emails to send CC and BCC. For example, hyperlink `"mailto:sales@perpetuumsoft.com?subject=Suggestion&Body=Hello!%0DI%20have%20some%20ideas%20for%20you.&bcc=support@perpetuumsoft.com"` opens email client with the following data:



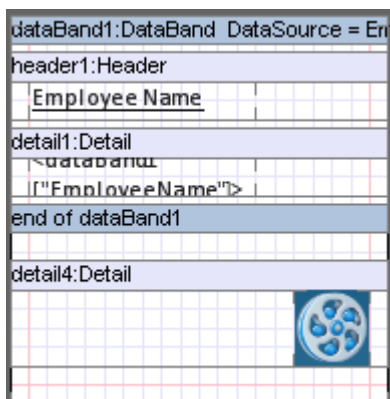
Hello!  
I have some ideas for you. |

### Bookmarks

To make navigation through the report easier, it is possible to use Bookmark property. This property creates a bookmark that allows you to go to this object from any place of the report with

the help of Content tree in Report Viewer. Name of the bookmark should start with # symbol, otherwise Bookmark property value will be ignored.

Report Template:

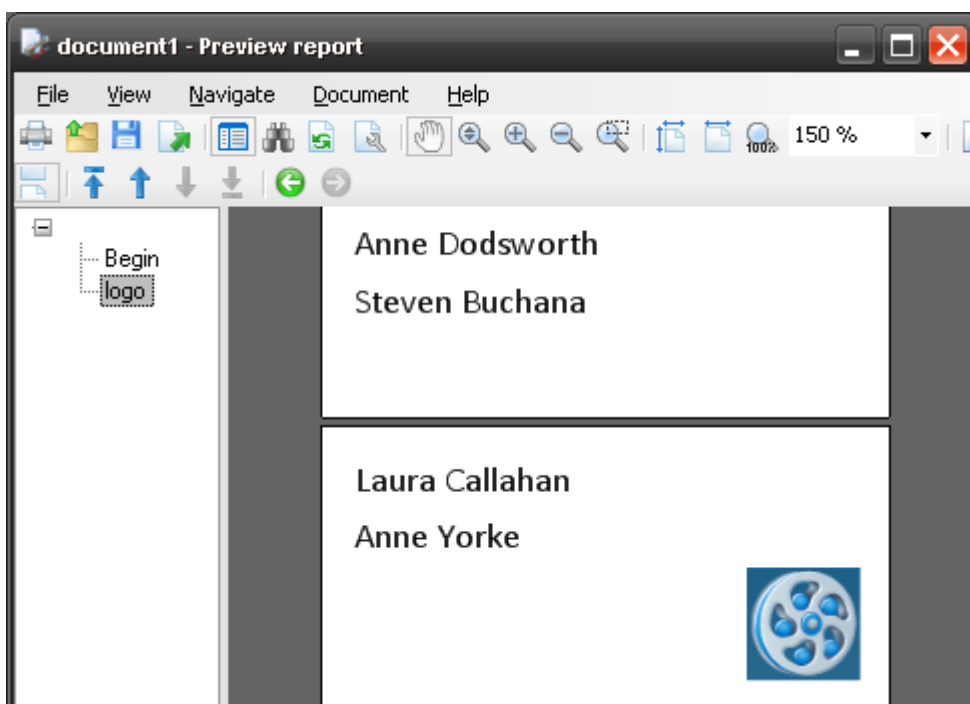


```
textBoxHeader.Bookmark = #Begin;
```

```
picture.Bookmark = #logo;
```



Report in the Report Viewer:

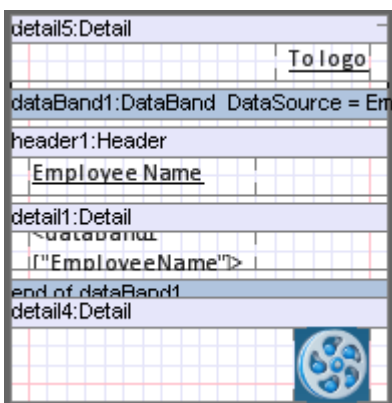


Bookmark tree is located on the left. "Show content" button is used to display/hide the tree. When you select some item in the Content area, report area containing element for which the corresponding bookmark is set will be opened.

### *Joint Use of Hyperlinks and Bookmarks for Report Navigation*

Bookmarks can be used as to go to them via hyperlinks. It is necessary to set bookmark name in the Hyperlink property.

Report Template:

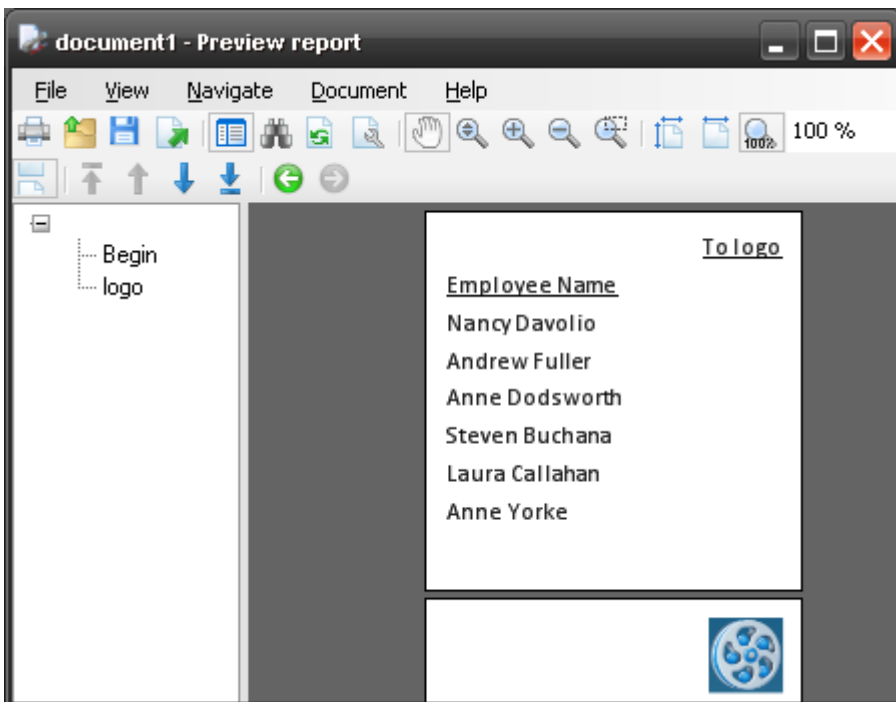


picture.Bookmark = #logo;

textBoxToLogo.Hyperlink = #logo

Navigation	
Bookmark	
Hyperlink	#logo

Report in the Report Viewer:



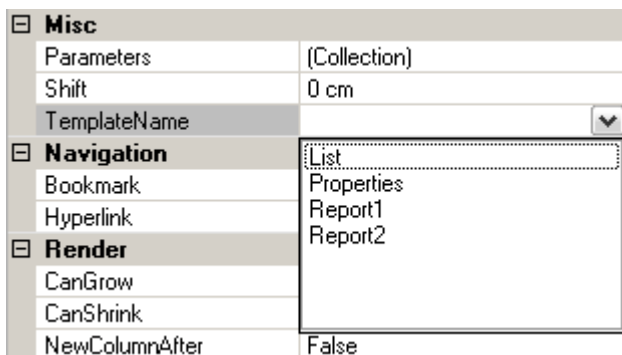
When you click on the TextBox element area with the text "To logo" report part containing logo will open.

To get back to the previous report area, it is possible to use "Back" hyperlink.

## Using subreports

Mechanism of using subreports is used when it is necessary to join several reports in a single one. SubReport is used for this purpose. It defines included subreport and its location in the template. Subreport template should be located in the ReportManager current template is located in.

To set subreport template, TemplateName property of the SubReport is used. Property combo box contains report names contained in the current ReportManager (if ReportName property for ReportSlot is not set in the ReportManager, template is not displayed in the list).

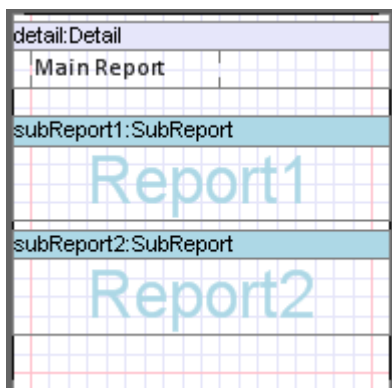


Template of master report	Subreport template	Report
<p>Property subReport.TemplateName = Report1</p>		

Final report contains only those elements that are located inside bands (DataBand, Detail etc.). If style is not set for subreport elements, style of the master template or SubReport band is applied to them.

In order to set shift from the left report border Shift property of the SubReport is used.

Report Template:



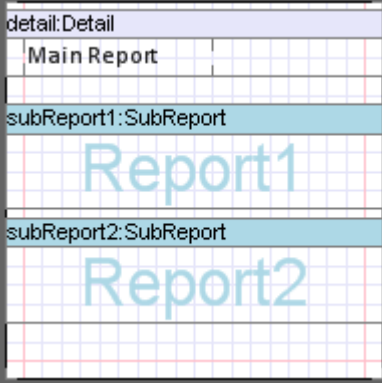
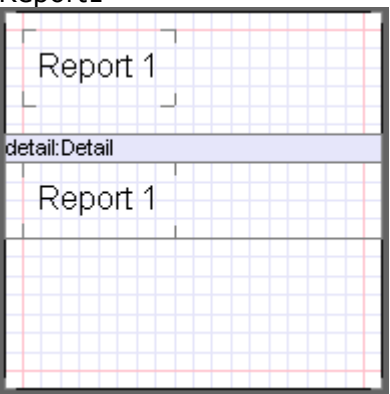
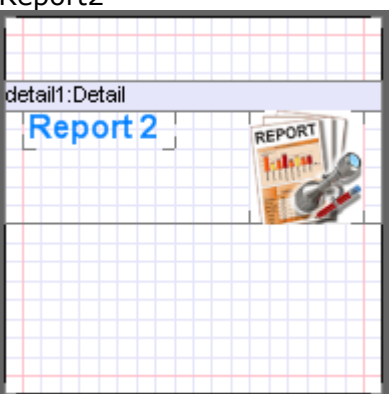



Property	Report										
<table border="1"><tr><td colspan="2"><input type="checkbox"/> <b>Misc</b></td></tr><tr><td>Parameters</td><td>(Collection)</td></tr><tr><td>Shift</td><td>0 cm</td></tr><tr><td>TemplateName</td><td><b>Report1</b></td></tr></table>	<input type="checkbox"/> <b>Misc</b>		Parameters	(Collection)	Shift	0 cm	TemplateName	<b>Report1</b>	<table border="1"><tr><td>Main Report</td></tr><tr><td>Report 1</td></tr></table>	Main Report	Report 1
<input type="checkbox"/> <b>Misc</b>											
Parameters	(Collection)										
Shift	0 cm										
TemplateName	<b>Report1</b>										
Main Report											
Report 1											
<table border="1"><tr><td colspan="2"><input type="checkbox"/> <b>Misc</b></td></tr><tr><td>Parameters</td><td>(Collection)</td></tr><tr><td>Shift</td><td><b>2 cm</b></td></tr><tr><td>TemplateName</td><td><b>Report1</b></td></tr></table>	<input type="checkbox"/> <b>Misc</b>		Parameters	(Collection)	Shift	<b>2 cm</b>	TemplateName	<b>Report1</b>	<table border="1"><tr><td>Main Report</td></tr><tr><td>Report 1</td></tr></table>	Main Report	Report 1
<input type="checkbox"/> <b>Misc</b>											
Parameters	(Collection)										
Shift	<b>2 cm</b>										
TemplateName	<b>Report1</b>										
Main Report											
Report 1											



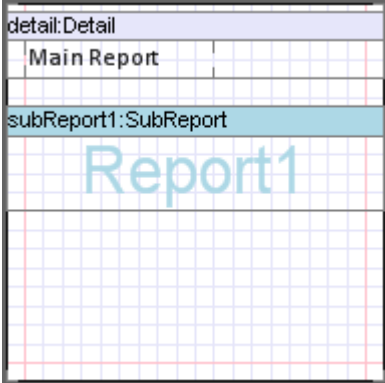
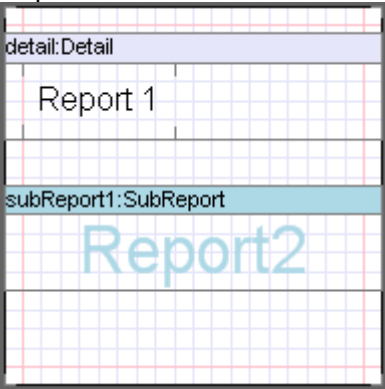
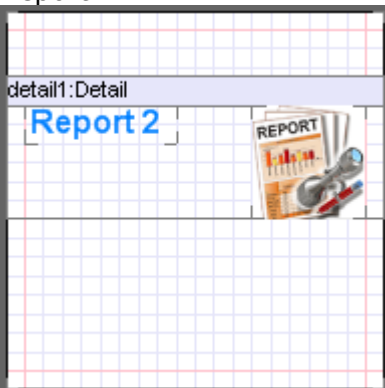

### Complex Report

Any amount of subreports can be added to a single report.

Master report template	Subreport template	Report
 <p>Properties: subReport1.TemplateName = Report1 subReport2.TemplateName = Report2</p>	<p>Report1</p>  <p>Report2</p> 	<p>Main Report</p> <p>Report 1</p> <p><b>Report 2</b></p> 



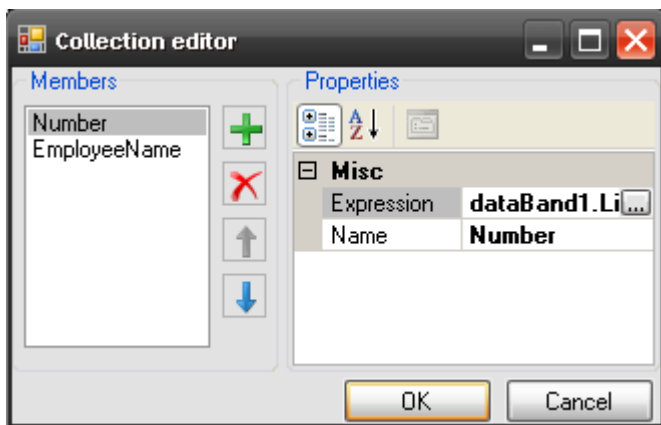
It is possible to use nested reports.

Master report template	Subreport template	Report
 <p>Property: subReport1.TemplateName = Report1</p>	<p>Report1</p>  <p>Property subReport1.TemplateName = Report2</p> <p>Report2</p> 	 <p>Main Report Report 1 Report 2</p>



### Passing Parameters to Subreport

Parameters property of the SubReport can be used to pass parameters from master report to subreport. Collection Editor is editor of this property.

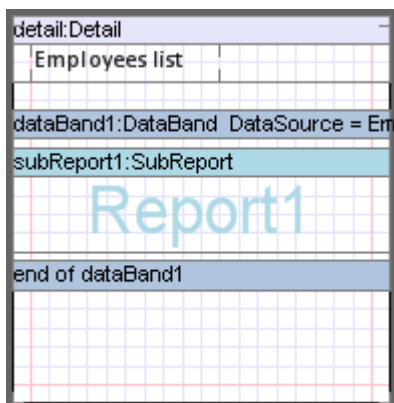


Expression property sets expression which calculation result will be passed as a parameter. Name property sets name used to refer to parameter value in subreport.

To get parameter value in subreport `GetParameter("<ParameterName>")` expression is used.

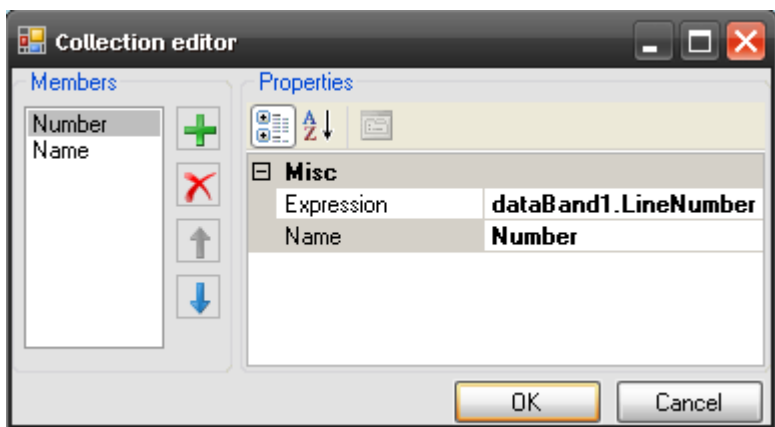
Sample of a template displaying employees list. Names and sequence numbers are displayed in subreport

Master Report Template:

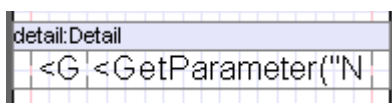


Property `subReport1.TemplateName = Report1`

Two parameters are added to the `subReport1.Parameters` collection. First parameter: Expression = `dataBand1.LineNumber`, Name = `Number`. Second parameter: Expression = `dataBand1["EmployeeName"]`. Name = `Name`.



Subreport Template:



```
textBox1.Value = GetParameter("Number");
```

```
textBox2.Value = GetParameter("Name")
```

Report:

Employees list	
1	Nancy Davolio
2	Andrew Fuller
3	Anne Dodsworth
4	Steven Buchana
5	Laura Callahan
6	Anne Yorke



## Zip code

To display zip code ZipCode element is used. Code is set in the Code property.

Appearance	
Angle	0
Border	
Code	22550
Color	Black

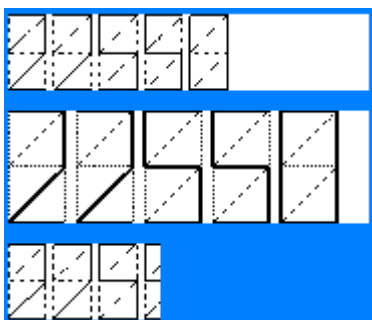
Code color is set in the Color property.

Appearance	
Angle	0
Border	
Code	22550
Color	DodgerBlue
Fill	[None]

Element in the template:



Size of the element is set in the Size property. It is necessary to consider that proportion of zip code is constant. Height of the code label depends on the element height; if label doesn't fit element width, code is cut.



## Master-Report

There is an ability to use master report. When the document using master template is generated, final document includes results of rendering of master template and document template. It allows you to create group of homogeneous reports.

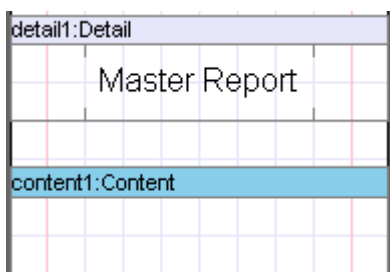
To set master report, it is necessary to set its name in the Document.MasterReport property.



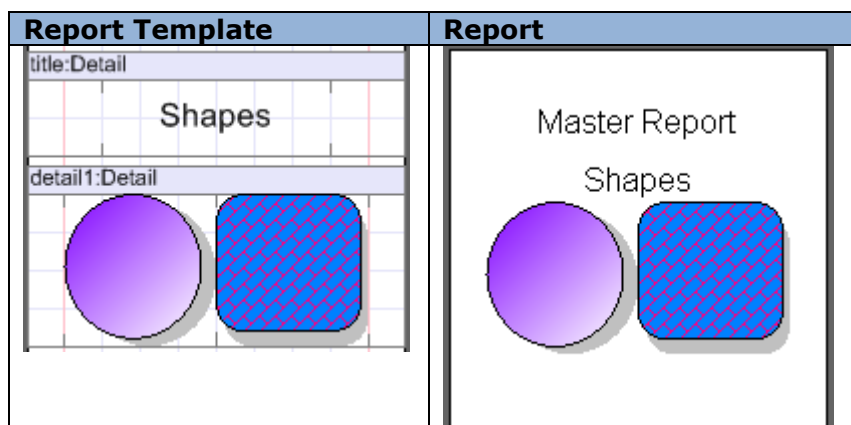
During rendering, template with the set name will be got through the IResolveSubReport interface. For example, master template with necessary name can be added to the same ReportManager initial template is added to.

When you create template used as a master one, it is necessary to add Content element to this template.

Master report template:



Generation of a document using master template is executed in the following way. Resulting report includes result of rendering of master template and Content element is substituted by the result of the initial report.





### Creation of Reports Using Wizard

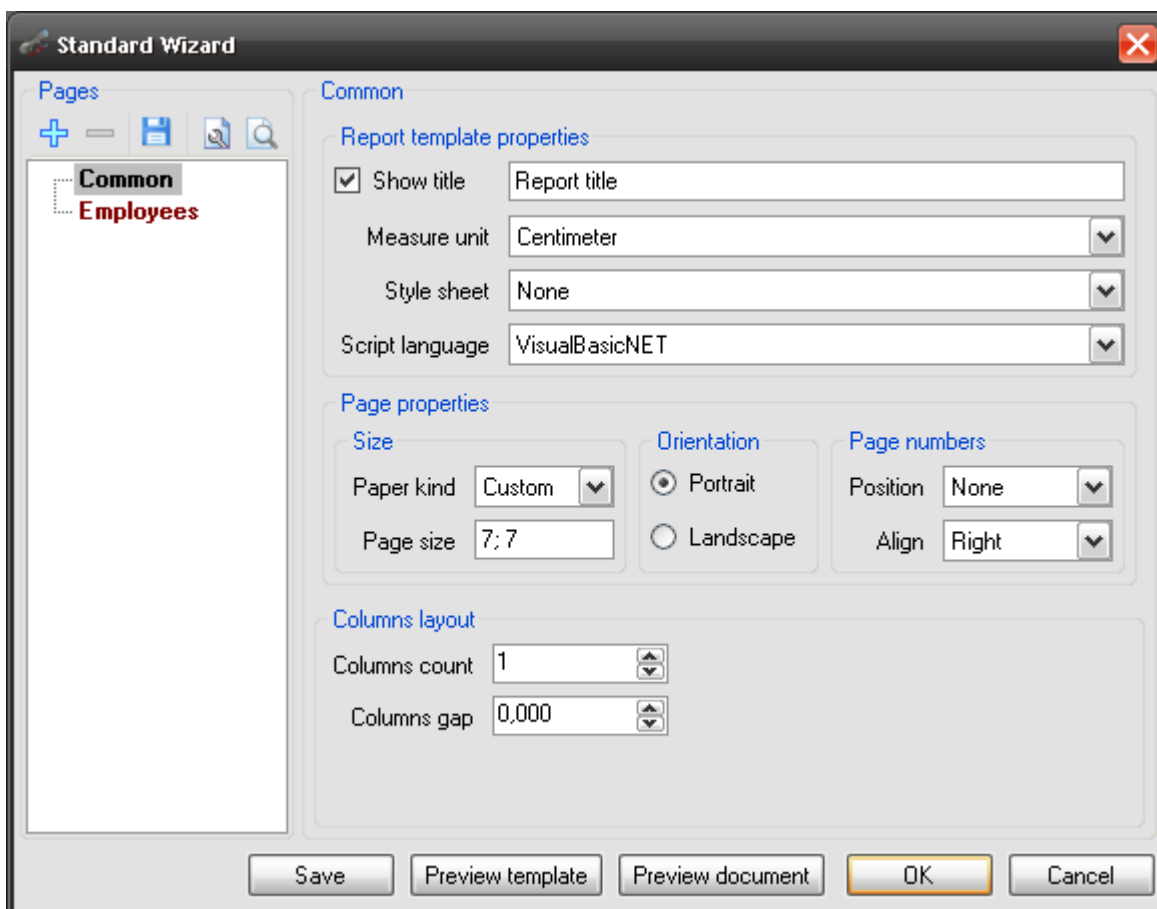
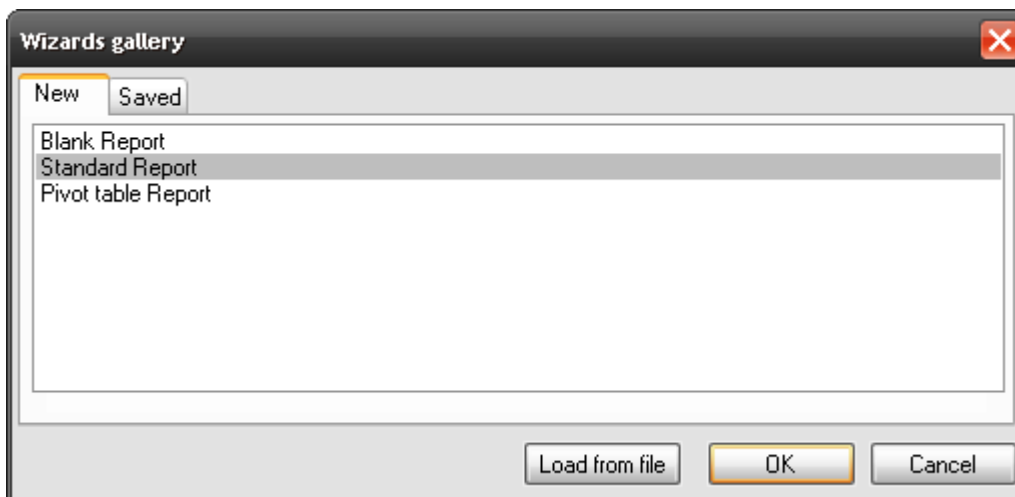
There is an easy way to create simple report templates - Wizards. They allow the setting of report parameters: appearance, data sources; and then the building of the report template by set parameters automatically.

It is possible to save all settings made in Wizard to the .rsw file into the xml format.

#### Creation of simple reports (Standard Wizard)

Standard Wizard allows the creation of report template in which the data from the data source is output as a list or cards.

It is possible to open Standard Wizard in the Wizards Gallery dialog.



### Report Properties in Standard Wizard

It is possible to set template common properties, which define the report appearance, in the Common section.

The report name, set in the Title field using the Document.Title property, will be displayed in the report header:

Title	Report Template	Report				
<p>Report template properties</p> <p><input checked="" type="checkbox"/> Show title Employees</p> <p>Measure unit Centimeter</p>		<p>Employees</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nancy</td> <td>(206)555</td> </tr> </tbody> </table>	Name	Phone	Nancy	(206)555
Name	Phone					
Nancy	(206)555					

The report measure units are set with Measure unit field (the Document.MetricUnit property).

Report style can be set in the Style sheet (the Document.StyleSheet property).

Style sheet	Property	Report														
<p>Measure unit Centimeter</p> <p>Style sheet Modern1</p> <p>Script language VisualBasic.NET</p>	<p><input type="checkbox"/> Render StyleSheet Modern1</p> <p><input type="checkbox"/> Script</p>	<p>Employees</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nancy Davolio</td> <td>(206)555-9857</td> </tr> <tr> <td>Andrew Fuller</td> <td>(206) 555-9482</td> </tr> <tr> <td>Anne Dodsworth</td> <td>(71) 555-4444</td> </tr> <tr> <td>Steven Buchana</td> <td>(71) 555-4444</td> </tr> <tr> <td>Laura Callahan</td> <td>(71) 555-4444</td> </tr> <tr> <td>Anne Yorke</td> <td>(71) 555-4444</td> </tr> </tbody> </table>	Name	Phone	Nancy Davolio	(206)555-9857	Andrew Fuller	(206) 555-9482	Anne Dodsworth	(71) 555-4444	Steven Buchana	(71) 555-4444	Laura Callahan	(71) 555-4444	Anne Yorke	(71) 555-4444
Name	Phone															
Nancy Davolio	(206)555-9857															
Andrew Fuller	(206) 555-9482															
Anne Dodsworth	(71) 555-4444															
Steven Buchana	(71) 555-4444															
Laura Callahan	(71) 555-4444															
Anne Yorke	(71) 555-4444															
<p>Measure unit Centimeter</p> <p>Style sheet Modern</p> <p>Script language VisualBasic.NET</p>	<p><input type="checkbox"/> Render StyleSheet Modern</p> <p><input type="checkbox"/> Script</p>	<p>Employees</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Phone</th> </tr> </thead> <tbody> <tr> <td>Nancy Davolio</td> <td>(206)555-9857</td> </tr> <tr> <td>Andrew Fuller</td> <td>(206) 555-9482</td> </tr> <tr> <td>Anne Dodsworth</td> <td>(71) 555-4444</td> </tr> <tr> <td>Steven Buchana</td> <td>(71) 555-4444</td> </tr> <tr> <td>Laura Callahan</td> <td>(71) 555-4444</td> </tr> <tr> <td>Anne Yorke</td> <td>(71) 555-4444</td> </tr> </tbody> </table>	Name	Phone	Nancy Davolio	(206)555-9857	Andrew Fuller	(206) 555-9482	Anne Dodsworth	(71) 555-4444	Steven Buchana	(71) 555-4444	Laura Callahan	(71) 555-4444	Anne Yorke	(71) 555-4444
Name	Phone															
Nancy Davolio	(206)555-9857															
Andrew Fuller	(206) 555-9482															
Anne Dodsworth	(71) 555-4444															
Steven Buchana	(71) 555-4444															
Laura Callahan	(71) 555-4444															
Anne Yorke	(71) 555-4444															

The Script Language field allows the selection of script language. Script language is set using the Document.ScriptLanguage property.

### Page Properties in Standard Wizard

The Page Properties section allows the setting of page properties. Paper format is selected from Paper kind list and is set by the Page.PaperKind property. In order to set arbitrary page size you should select Custom in Paper kind combo box and set the value to the Page size field.

Size

Paper kind Custom

Page size 7; 7

It is possible to set landscape or portrait page orientation in the Orientation section.

Orientation

Portrait

Landscape



Headlines are used to output page numbers. It is possible to set display mode and text alignment of headliners in the Page Number section.

Page numbers

Position: Both

Align: Right

Report:

7

## Employees

Name	Phone
Nancy Davolio	(206)555-9857
Andrew Fuller	(206) 555-9482
Anne Dodsworth	(71) 555-4444
Steven Buchana	(71) 555-4444
Laura Callahan	(71) 555-4444

7

It is possible to create multicolumn reports using Standard Wizard. To do this you should use Custom layout settings. Columns count defines the columns number, Columns gap defines the gap between columns (it is set with the DataBand.ColumnsCount and DataBand.ColumnsGap properties).

Columns layout

Columns count: 2

Columns gap: 0,5

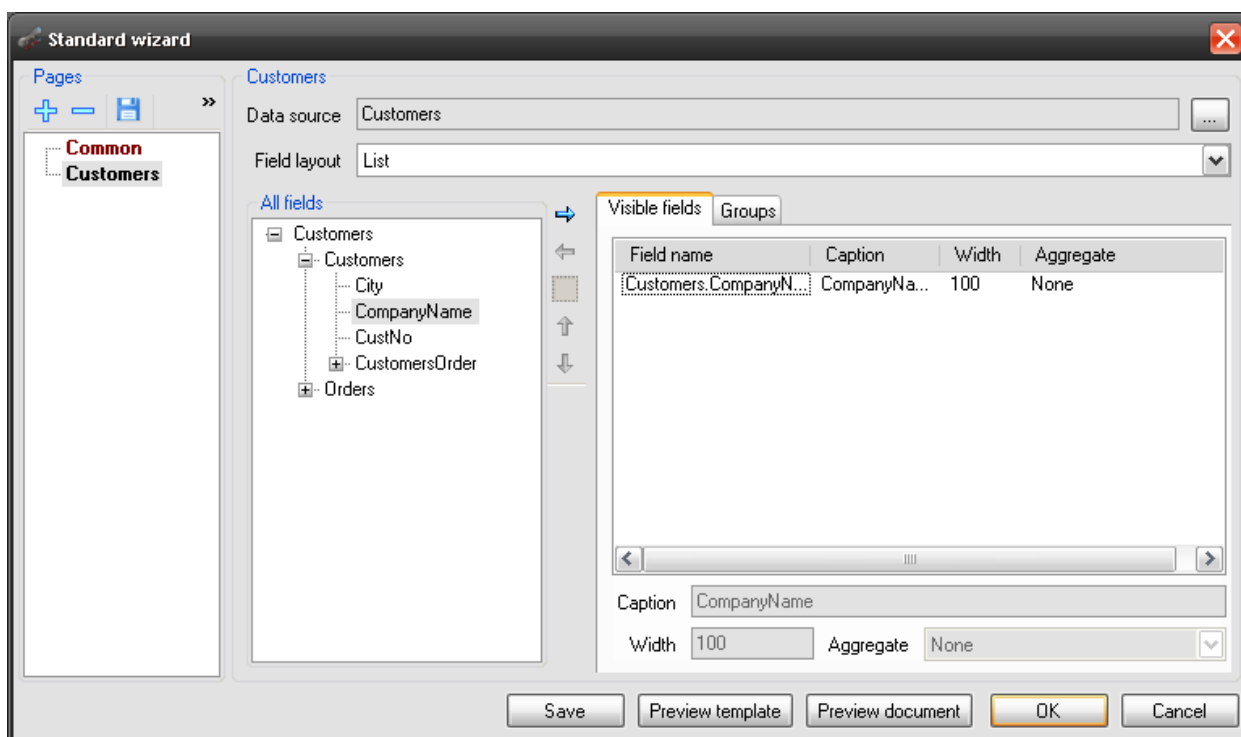
Report:

## Employees

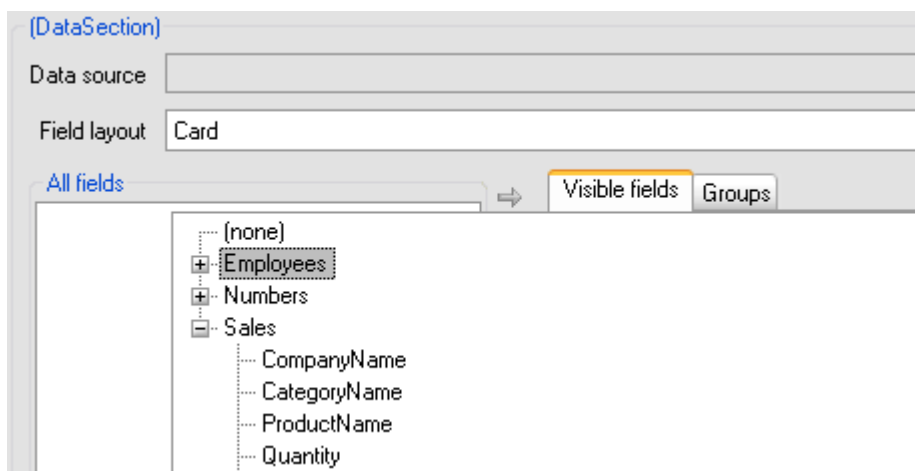
Nancy Davolio	Laura Callahan
Andrew Fuller	Anne Yorke
Anne Dodsworth	
Steven Buchana	

### Report Data for Simple List

The data for the report creation should be provided by the application, from which Report Designer is invoked. To add data you should press the "Add data" button. After this a new record in the template structure tree will appear and the section for data source setting and data display will open.

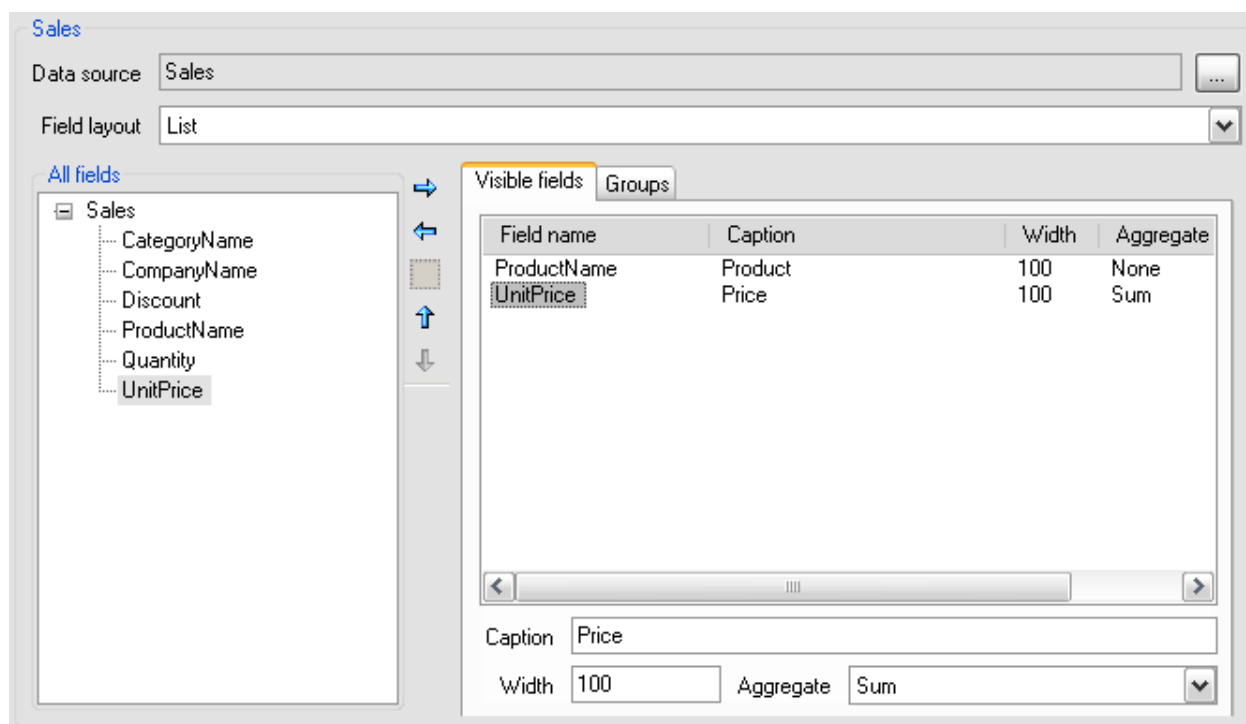


Data source is set in the Data Source field.



The fields of the data source, which is selected in the Data Source field, are displayed in All Fields list. It is possible to add field from the list to the table on the Visible tab using control buttons. It is possible to edit title, width and set aggregation function for each field.





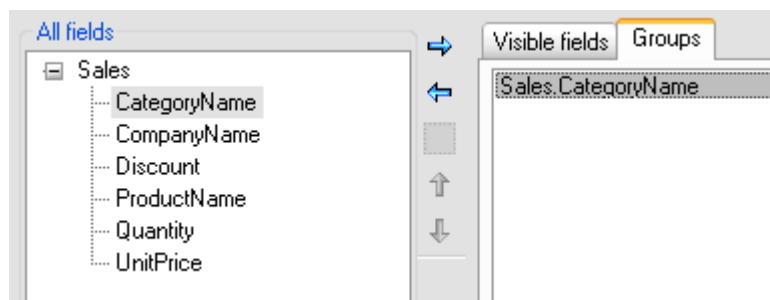
Report Template	Report																								
<pre>dataBand1:DataBand DataSource = Sales header1:Header   Product      Price detail1:Detail   &lt;GetData    &lt;GetData   ({"Sales.ProductName"}) ({"Sales.UnitPrice"}) footer1:Footer   &lt;GetTotal   ("Sales.UnitPrice") end of dataBand1</pre>	<table border="1"> <thead> <tr> <th>Product</th> <th>Price</th> </tr> </thead> <tbody> <tr><td>Chai</td><td>35,5</td></tr> <tr><td>Stout</td><td>105</td></tr> <tr><td>Maxilaku</td><td>2,6</td></tr> <tr><td>Geitost</td><td>15,8</td></tr> <tr><td>Flotemysost</td><td>240</td></tr> <tr><td>Courdavault</td><td>62,5</td></tr> <tr><td>Chai</td><td>35,5</td></tr> <tr><td>Ipoh Coffee</td><td>50,5</td></tr> <tr><td>Filo Mix</td><td>14</td></tr> <tr><td>Filo Mix</td><td>14</td></tr> <tr><td colspan="2" style="text-align: right;"><b>575,4</b></td></tr> </tbody> </table>	Product	Price	Chai	35,5	Stout	105	Maxilaku	2,6	Geitost	15,8	Flotemysost	240	Courdavault	62,5	Chai	35,5	Ipoh Coffee	50,5	Filo Mix	14	Filo Mix	14	<b>575,4</b>	
Product	Price																								
Chai	35,5																								
Stout	105																								
Maxilaku	2,6																								
Geitost	15,8																								
Flotemysost	240																								
Courdavault	62,5																								
Chai	35,5																								
Ipoh Coffee	50,5																								
Filo Mix	14																								
Filo Mix	14																								
<b>575,4</b>																									

Field layout list allows the selection of data view in the report: in the form of a list or cards.

Field layout	Report Template	Report																				
<pre>Field layout List</pre>	<pre>dataBand1:DataBand DataSource = Employees header1:Header   Employee      Phone detail1:Detail   &lt;GetData      &lt;GetData   ({"Employees.EmployeeName"}) ({"Employees.HomePhone"}) end of dataBand1</pre>	<table border="1"> <thead> <tr> <th>Employee</th> <th>Phone</th> </tr> </thead> <tbody> <tr><td>Nancy Davolio</td><td>(206)555-9857</td></tr> <tr><td>Andrew Fuller</td><td>(206) 555-9482</td></tr> <tr><td>Anne Dodsworth</td><td>(71) 555-4444</td></tr> <tr><td>Steven Buchana</td><td>(71) 555-4444</td></tr> <tr><td>Laura Callahan</td><td>(71) 555-4444</td></tr> <tr><td>Anne Yorke</td><td>(71) 555-4444</td></tr> </tbody> </table>	Employee	Phone	Nancy Davolio	(206)555-9857	Andrew Fuller	(206) 555-9482	Anne Dodsworth	(71) 555-4444	Steven Buchana	(71) 555-4444	Laura Callahan	(71) 555-4444	Anne Yorke	(71) 555-4444						
Employee	Phone																					
Nancy Davolio	(206)555-9857																					
Andrew Fuller	(206) 555-9482																					
Anne Dodsworth	(71) 555-4444																					
Steven Buchana	(71) 555-4444																					
Laura Callahan	(71) 555-4444																					
Anne Yorke	(71) 555-4444																					
<pre>Field layout Card</pre>	<pre>dataBand1:DataBand DataSource = Employees detail1_Employees_EmployeeName:Detail   Employee      &lt;GetData   ({"Employees.EmployeeName"}) detail1_Employees_HomePhone:Detail   Phone         &lt;GetData   ({"Employees.HomePhone"}) end of dataBand1</pre>	<table border="1"> <tbody> <tr><td>Employee</td><td>Nancy Davolio</td></tr> <tr><td>Phone</td><td>(206)555-9857</td></tr> <tr><td>Employee</td><td>Andrew Fuller</td></tr> <tr><td>Phone</td><td>(206) 555-9482</td></tr> <tr><td>Employee</td><td>Anne Dodsworth</td></tr> <tr><td>Phone</td><td>(71) 555-4444</td></tr> <tr><td>Employee</td><td>Steven Buchana</td></tr> <tr><td>Phone</td><td>(71) 555-4444</td></tr> <tr><td>Employee</td><td>Laura Callahan</td></tr> <tr><td>Phone</td><td>(71) 555-4444</td></tr> </tbody> </table>	Employee	Nancy Davolio	Phone	(206)555-9857	Employee	Andrew Fuller	Phone	(206) 555-9482	Employee	Anne Dodsworth	Phone	(71) 555-4444	Employee	Steven Buchana	Phone	(71) 555-4444	Employee	Laura Callahan	Phone	(71) 555-4444
Employee	Nancy Davolio																					
Phone	(206)555-9857																					
Employee	Andrew Fuller																					
Phone	(206) 555-9482																					
Employee	Anne Dodsworth																					
Phone	(71) 555-4444																					
Employee	Steven Buchana																					
Phone	(71) 555-4444																					
Employee	Laura Callahan																					
Phone	(71) 555-4444																					

### Grouping data

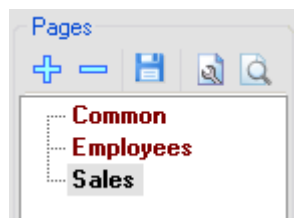
For creation of reports with grouping it is needed to transfer the grouping field to the table on the Groups tab from All fields tree.



Report Template	Report
dataBand1:DataBand DataSource = Sales	<b>Product</b> <b>Price</b>
header1:Header	<b>Beverages</b>
Product      Price	Chai      35,5
groupBand1:GroupBand Group = GetData("Sales.CategoryName")	Stout      105
header11:Header	Chai      35,5
<GetData("Sales.CategoryName")>	Ipoh Coffee      50,5
detail1:Detail	<b>226,5</b>
<GetData("Sales.ProductName")>      <GetData("Sales.UnitPrice")>	<b>Confections</b>
footer1:Footer	Maxilaku      2,6
<GetTotal("Sales.UnitPrice")>	<b>2,6</b>
end of groupBand1	<b>Dairy Products</b>
footer11:Footer	Geitost      15,8
<GetTotal("Sales.UnitPrice")>	Flotemysost      240
end of dataBand1	Courdavault      62,5
	<b>318,3</b>
	<b>Grains/Cereals</b>
	Filo Mix      14
	Filo Mix      14
	<b>28</b>
	<b>575,4</b>

### Several Data Sources

Data from several data sources can be output in the report sequentially. To do this you should add data source to the template structure tree and transfer the needed fields to Visible fields.

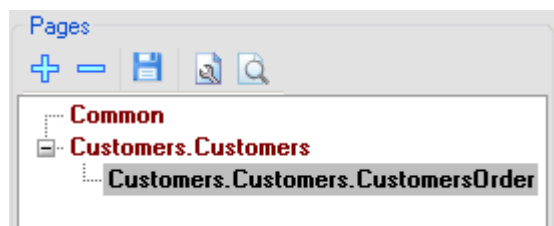


Report Template	Report
dataBand1:DataBand DataSource = Employees	<b>Employee</b> <b>Phone</b>
header1:Header	Nancy Davolio      (206)555-9857
Employee      Phone	Andrew Fuller      (206) 555-9482
detail1:Detail	Anne Dodsworth      (71) 555-4444
<GetData("Employees.EmployeeName")>      <GetData("Employees.HomePhone")>	Steven Buchana      (71) 555-4444
end of dataBand1	Laura Callahan      (71) 555-4444
	Anne Yorke      (71) 555-4444
	<b>Company</b> <b>Product</b>
	Futterkiste      Chai
	Futterkiste      Stout
	Futterkiste      Maxilaku
	Futterkiste      Geitost
dataBand2:DataBand DataSource = Sales	
header2:Header	
Company      Product	
detail2:Detail	
<GetData("Sales.CompanyName")>      <GetData("Sales.ProductName")>	
end of dataBand2	



### Master Detail Report

To create Master Detail reports it is necessary to add data source with details to the list of the other data source in the template structure tree.

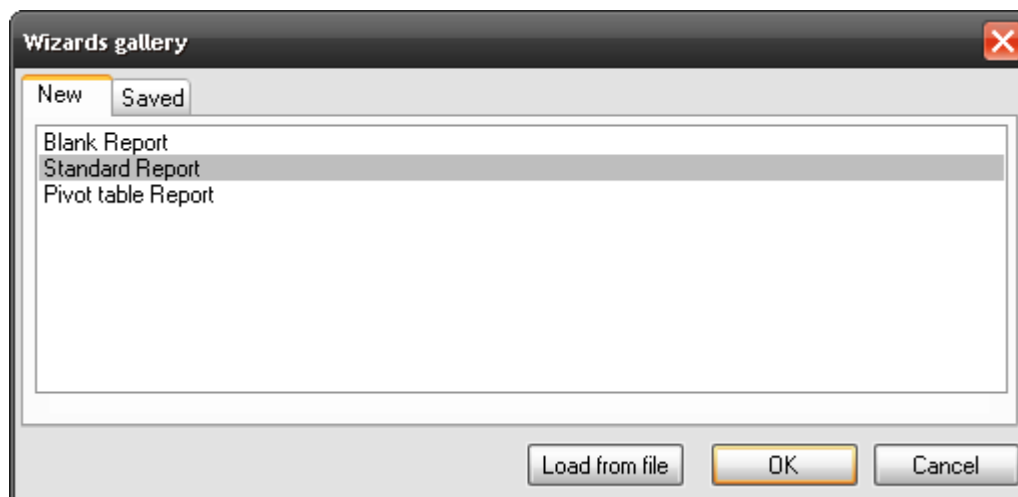


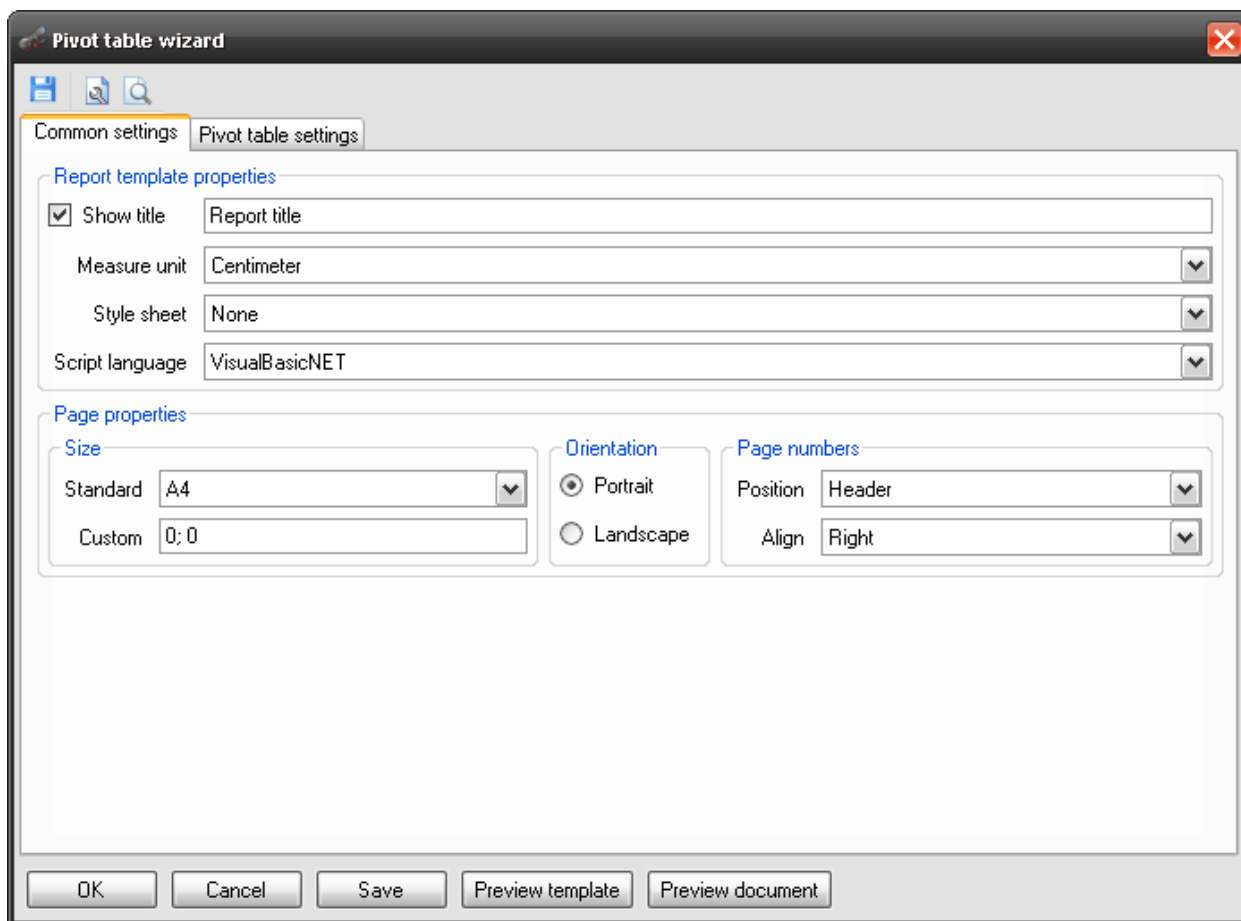
Report Template	Report
dataBand1:DataBand DataSource = Customers.Customers	Company Bon App'
detail1_Customers_Customers_CompanyName:Detail	Date Sum
Company <<GetData if("Customers.Custome	21.03.10 50.00
	15.02.10 14.50
dataBand1_1:DataBand DataSource = Customers.Customers.Custome	64,5
header1_1:Header	Company Chop-suey Chinese
Date Sum	Date Sum
	03.04.10 134.00
	15.01.10 45.45
detail1_1:Detail	01.02.10 500.00
<<GetData <<GetData if("Customers.C if("Customers.Custome	30.12.09 6.03
footer1_1:Footer	685,48
<<Get total if("Customers.Custom	Company Maison Dewey
	Date Sum
end of dataBand1_1	06.06.10 66.66
end of dataBand1	66,66

### Creation of Report with Pivot Table (Pivot Table Wizard)

Pivot Table Wizard allows the creation of report template which outputs data from the data source in the form of pivot table.

It is possible to open Pivot Table Wizard in the Wizards Gallery dialog.





### Report Properties in Pivot Table Wizard

It is possible to set template common properties, which define the report appearance, on the Common settings tab.

The report name, set in the Title field using the Document.Title property, will be displayed in the report header:

Title	Report Template	Report
<p>Report template properties</p> <p><input checked="" type="checkbox"/> Show title    Sales</p> <p>Measure unit    Centimeter</p>	<p>reportTitle:Detail</p> <p>&lt;Document.Title&gt;</p>	<p>Sales</p>

The report measure units are set with Measure unit field (the Document.MetricUnit property).

Report style can be set in the Style field:

Style Sheet	Report Template	Report																																																																													
<p>Style sheet    Modern1</p>	<p>reportTitle:Detail</p> <p>&lt;Document.Title&gt;</p> <p>pivotTable1:PivotTable</p> <table border="1"> <thead> <tr> <th colspan="2"></th> <th colspan="2">Company</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Unit</th> <th>Unit</th> </tr> </thead> <tbody> <tr> <td rowspan="3">Group</td> <td>Item 1</td> <td>0.0</td> <td>0.0</td> </tr> <tr> <td>Item 2</td> <td>0.0</td> <td>0.0</td> </tr> <tr> <td>Total</td> <td>0.0</td> <td>0.0</td> </tr> <tr> <td colspan="2">Total</td> <td>0.0</td> <td>0.0</td> </tr> </tbody> </table>			Company		Category	Product	Unit	Unit	Group	Item 1	0.0	0.0	Item 2	0.0	0.0	Total	0.0	0.0	Total		0.0	0.0	<p>Sales</p> <table border="1"> <thead> <tr> <th colspan="2"></th> <th colspan="3">Company</th> </tr> <tr> <th>Category</th> <th>Product</th> <th>Unit</th> <th>Unit</th> <th>Unit</th> </tr> </thead> <tbody> <tr> <td rowspan="3">Beverages</td> <td>Chai</td> <td>35.5</td> <td>35.5</td> <td></td> </tr> <tr> <td>Ipoh Coffee</td> <td></td> <td>50.5</td> <td></td> </tr> <tr> <td>Stout</td> <td>105</td> <td></td> <td></td> </tr> <tr> <td colspan="2">Total</td> <td>140.5</td> <td>86</td> <td></td> </tr> <tr> <td rowspan="2">Confections</td> <td>Maxolaku</td> <td>2.6</td> <td></td> <td></td> </tr> <tr> <td>Total</td> <td>2.6</td> <td></td> <td></td> </tr> <tr> <td rowspan="3">Dairy Products</td> <td>Courdavault</td> <td>62.5</td> <td></td> <td></td> </tr> <tr> <td>Flofemysost</td> <td>240</td> <td></td> <td></td> </tr> <tr> <td>Geitosi</td> <td>15.8</td> <td></td> <td></td> </tr> <tr> <td colspan="2">Total</td> <td>318.3</td> <td></td> <td></td> </tr> </tbody> </table>			Company			Category	Product	Unit	Unit	Unit	Beverages	Chai	35.5	35.5		Ipoh Coffee		50.5		Stout	105			Total		140.5	86		Confections	Maxolaku	2.6			Total	2.6			Dairy Products	Courdavault	62.5			Flofemysost	240			Geitosi	15.8			Total		318.3		
		Company																																																																													
Category	Product	Unit	Unit																																																																												
Group	Item 1	0.0	0.0																																																																												
	Item 2	0.0	0.0																																																																												
	Total	0.0	0.0																																																																												
Total		0.0	0.0																																																																												
		Company																																																																													
Category	Product	Unit	Unit	Unit																																																																											
Beverages	Chai	35.5	35.5																																																																												
	Ipoh Coffee		50.5																																																																												
	Stout	105																																																																													
Total		140.5	86																																																																												
Confections	Maxolaku	2.6																																																																													
	Total	2.6																																																																													
Dairy Products	Courdavault	62.5																																																																													
	Flofemysost	240																																																																													
	Geitosi	15.8																																																																													
Total		318.3																																																																													

The Script Language field allows the selection of the script language using the Document.ScriptLanguage property.

### Page Properties in Pivot Table Wizard

The Page Properties section allows the setting of page properties. Paper format is selected from Paper kind list and is set with the Page.PaperKind property. In order to set arbitrary page size you should select Custom in Paper kind combo box and set the value to the Page size field.

Size

Paper kind    A4

Page size    0; 0

It is possible to set landscape or portrait page orientation in the Orientation section.

Orientation

Portrait

Landscape

Headlines are used to output page numbers. It is possible to set display mode and text alignment of headliners in the Page Number section.

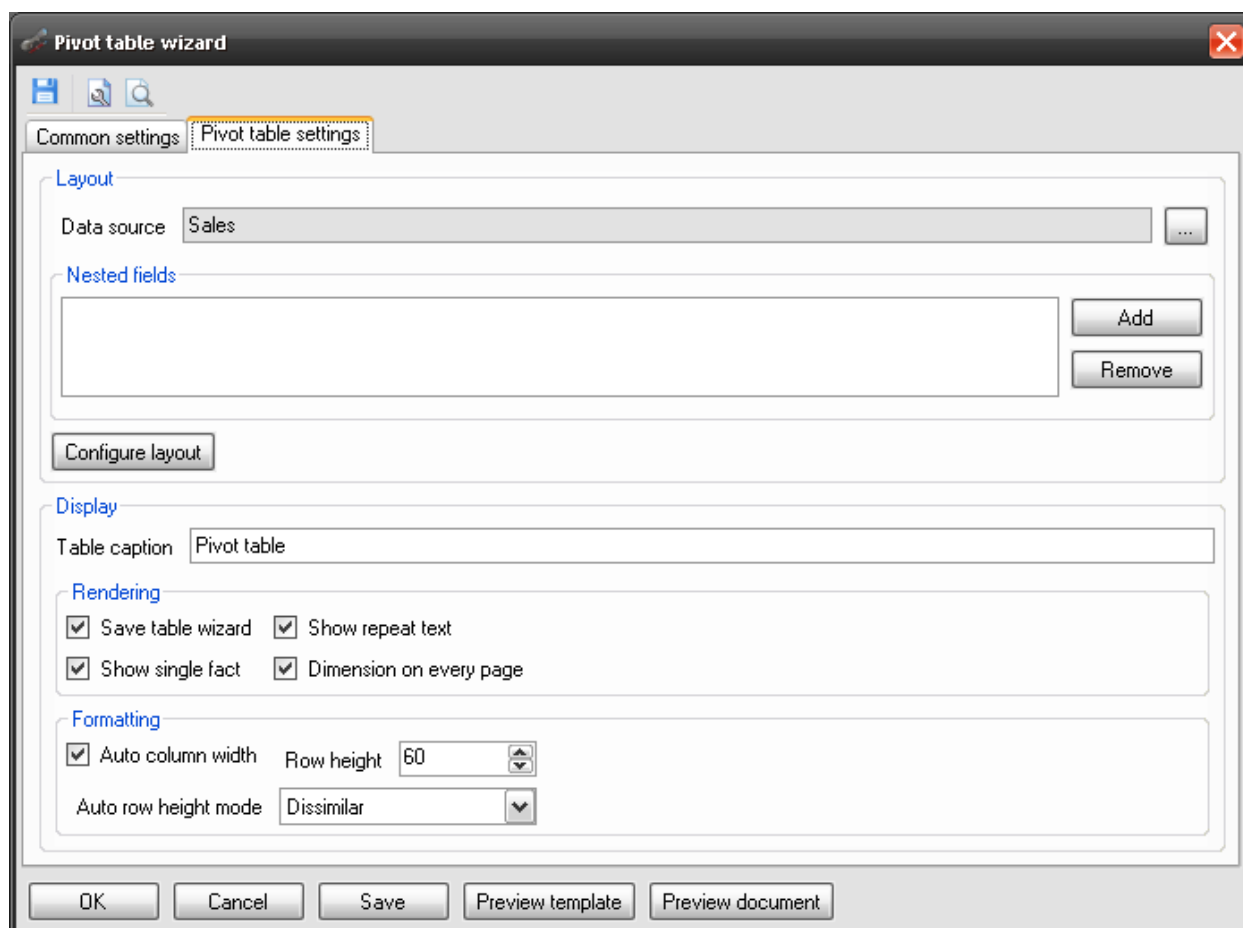
Page numbers

Position    Header

Align    Right

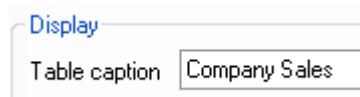
### Data for the Table

The data and table appearance are defined with the settings on the Pivotal table settings tab.



The data source is set in the Data source field. To adjust the pivot table cells you should open Source Fields editor with the "Configure layout" button. (More information about data in the pivot table)

It is possible to set pivot table caption in the Table caption field.



**Company Sales**

		Company			
		Futterkiste	Karkki Oy	Shopy	Total
Category	Product	Price	Price	Price	Price
Beverages	Chai	35.5	35.5		71
	Ipoh Coffee		50.5		50.5
	Stout	105			105
	Total	140.5	86		226.5
Confections	Maxilaku	2.6			2.6
	Total	2.6			2.6
Dairy Products	Courdavault	62.5			62.5
	Flotemysost	240			240
	Geitost	15.8			15.8
	Total	318.3			318.3

The Rendering group of properties allows the setting of such properties of the PivotTable as: ShowRepeatText, ShowSingleFact, and DementionOnEveryPage.



#### Rendering

- Save table wizard
- Show repeat text
- Show single fact
- Dimension on every page

The size of cells of the pivot table can be set in the Formatting section. It is possible to set the PivotTable.AutoColumnWidth, PivotTable.RowHeight and PivotTable.AutoRowHeight properties.

#### Formatting

- Auto column width
- Row height
- Auto row height mode

(More information about pivot table properties)



## Using Product Features

### Scripts

#### *Designation and Abilities*

Scripts are abstract declarator of .Net type connected to the document template. This type is called "DocumentScripts".

Some particular methods of the DocumentScript type are called during report generation. Most events in the generation process result in call of certain methods; logic of these methods is defined in scripts.

To make it simpler to understand, we can consider scripts as a set of event handlers initialized during generation of a report or separate object, data processing; scripts are used also for declaring secondary logic, for example, for handling mouse click on the Windows Forms.

Main designation of scripts is provision of data necessary for report generation, making decisions based on this data, managing rendering process.

Methods of the DocumentScripts class have access to:

- Data from document template data source;
- Object model of the document template (document, pages, objects);
- Document rendering engine (special object representing rendering process with properties stating current rendering status and methods influencing the further rendering process);
- All public data and methods of the application and assemblies the application refers to.

Scripts allow flexible management of report building. It is a .NET code that can use all .NET features, standard and third party libraries.

You can use C#, Visual Basic.Net or any other language supporting .Net environment as a script language.

Note: All scripts are saved as source code, that is why corresponding compiler should be installed on the client computer where report generation will be executed. C#, Visual Basic. NET and Jscript.NET compilers are included in the Microsoft .NET Framework.

#### *Scripting Background*

Inner mechanism of using scripts in the report generator. All scripts and expressions form class of the following structure:

```
public class Script : PerpetuumSoft.Reporting.Rendering.ReportScriptBase
{
    private PerpetuumSoft.Reporting.DOM.Page page1;
    private PerpetuumSoft.Reporting.DOM.TextBox textBox1;
    ...
    <Document.CommonScript>
    public Script(PerpetuumSoft.Reporting.DOM.Document document,
PerpetuumSoft.Reporting.Components.ObjectPointerCollection dataObjects,
PerpetuumSoft.Reporting.Rendering.RenderEngine engine) : base(document, dataObjects, engine)
    {
        this.page1 = ((PerpetuumSoft.Reporting.DOM.Page)(document.ControlByName("page1")));
        this.textBox1 = ((PerpetuumSoft.Reporting.DOM.TextBox)
(document.ControlByName("textBox1")));
        ...
        this.page1.ManualBuild += new System.EventHandler(this.page1_ManualBuild);
        this.textBox1.Generate += new System.EventHandler(this.textBox1_Generate);
        ...
    }

    private void page1_ManualBuild(object sender, System.EventArgs e)
```





```

{
    ...
}

private void textBox1_Generate(object sender, System.EventArgs e)
{
    this.textBox1.Value = <binding expression for Value property of the textBox1>
    ...
    <textBox1.GenerateScript>
}
...
}

```

Here is a real sample of resulting class that is generated for the second sample of calculating running total:

```

namespace PerpetuumSoft.Reporting.ReportScript
{
    #line 1 "Document$ImportsString"
    using System;
    #line default
    #line hidden
    #line 1 "Document$ImportsString"
    using PerpetuumSoft.Reporting.DOM;
    #line default
    #line hidden
    #line 1 "Document$ImportsString"
    using PerpetuumSoft.Framework.Drawing;
    #line default
    #line hidden
    public class Script : PerpetuumSoft.Reporting.Rendering.ReportScriptBase
    {
        private PerpetuumSoft.Reporting.DOM.TextBox textBox2;
        private PerpetuumSoft.Reporting.DOM.TextBox textBox3;
        private PerpetuumSoft.Reporting.DOM.TextBox textBox1;
        private PerpetuumSoft.Reporting.DOM.Header header1;
        private PerpetuumSoft.Reporting.DOM.Detail detail1;
        private PerpetuumSoft.Reporting.DOM.TextBox textBox4;
        private PerpetuumSoft.Reporting.DOM.DataBand dataBand1;
        private PerpetuumSoft.Reporting.DOM.Page page1;
    #line 1 "Document$Common"
        int sum1 = 0;
        int sum2 = 0;
    #line default
    #line hidden
        public Script(PerpetuumSoft.Reporting.DOM.Document document,
PerpetuumSoft.Reporting.Components.ObjectPointerCollection dataObjects,
PerpetuumSoft.Reporting.Rendering.RenderEngine engine)
            : base(document, dataObjects, engine)
        {
            this.textBox2 =
((PerpetuumSoft.Reporting.DOM.TextBox)(document.ControlByName("textBox2")));
            this.textBox3 =
((PerpetuumSoft.Reporting.DOM.TextBox)(document.ControlByName("textBox3")));
            this.textBox1 =
((PerpetuumSoft.Reporting.DOM.TextBox)(document.ControlByName("textBox1")));
            this.header1 =
((PerpetuumSoft.Reporting.DOM.Header)(document.ControlByName("header1")));
            this.detail1 =
((PerpetuumSoft.Reporting.DOM.Detail)(document.ControlByName("detail1")));
            this.textBox4 =
((PerpetuumSoft.Reporting.DOM.TextBox)(document.ControlByName("textBox4")));
            this.dataBand1 =
((PerpetuumSoft.Reporting.DOM.DataBand)(document.ControlByName("dataBand1")));

```

```

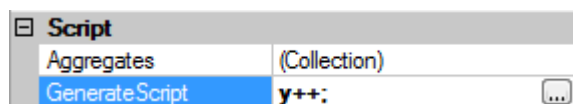
        this.page1 =
((PerpetuumSoft.Reporting.DOM.Page)(document.ControlByName("page1")));
        this.textBox2.Generate += new System.EventHandler(this.textBox2_Generate);
        this.textBox1.Generate += new System.EventHandler(this.textBox1_Generate);
        this.dataBand1.Generate += new System.EventHandler(this.dataBand1_Generate);
    }
    private void textBox2_Generate(object sender, System.EventArgs e)
    {
#line 1 "textBox2$Value"
        this.textBox2.Value = sum1 - sum2 + (int)dataBand1.DataItem;
#line default
#line hidden
#line 1 "textBox2$Generate"
#line default
#line hidden
    }
    private void textBox1_Generate(object sender, System.EventArgs e)
    {
#line 1 "textBox1$Value"
        this.textBox1.Value = dataBand1.DataItem;
#line default
#line hidden
#line 1 "textBox1$Generate"
#line default
#line hidden
    }
    private void dataBand1_Generate(object sender, System.EventArgs e)
    {
#line 1 "dataBand1$Generate"
        if (!Engine.IsDoublePass)
            sum1 += (int)dataBand1.DataItem;
        else
            sum2 += (int)dataBand1.DataItem;
#line default
#line hidden
    }
}

```

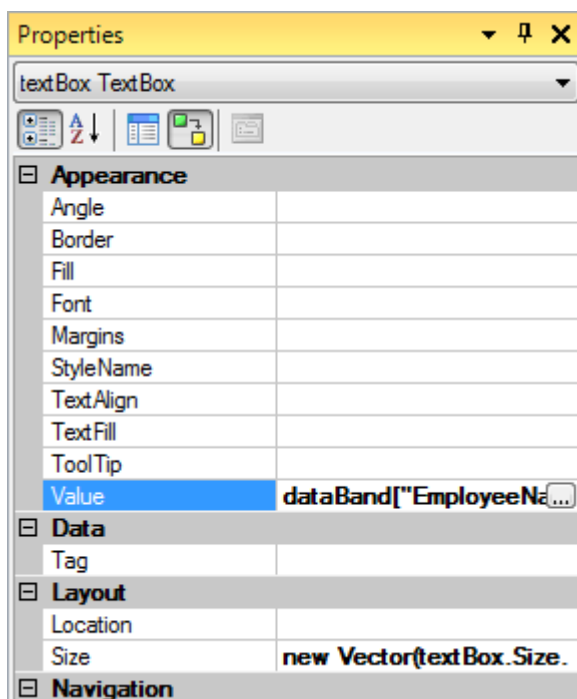
### Ways of Setting Scripts

Scripts are set and edited in the template designer and are stored together with the template. They are stored as text in string properties of the object and are available in the Property Grid.

All template objects have properties of the "Script" group; there you can set scripts of various types.



Scripts of a special type Property Binding are set on the "Bindings" tab of the Property Grid.



To get immediate access to Property Bindings you can use special elements on the Binding bar.

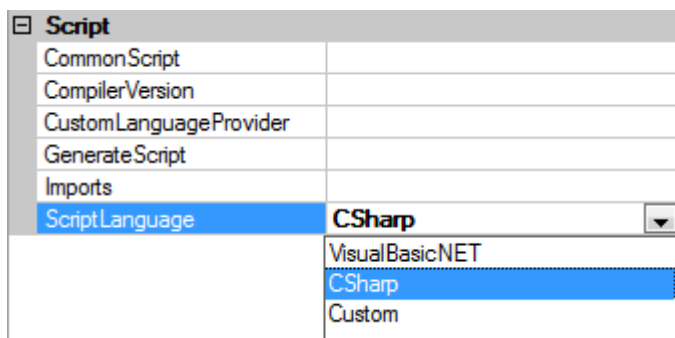


There is a special Script Editor with syntax highlighting and code completion designed for scripts editing.

### Setting Script Language and Compiler Version

It is possible to select VB.NET, C# or any other .NET programming language for writing scripts (J#, managed c++, any other).

Language is set in the Document.ScriptLanguage property. VisualBasic.NET is a default language.



In order to use any other language, it is necessary to specify location of the required compiler. It should be name of the System.CodeDom.Compiler.CodeDomProvider descendant type.

If necessary, it is possible to set compiler version using the Document.CompilerVersion property. For example, to use C# v. 3.5, it is necessary to set Document.CompilerVersion = "3.5".

Note: Scripts are compiled at the moment the report is generated, that is why compiler of the corresponding version should be installed on every computer where the reports will be generated by this template.


### Script Types

Scripts can be divided in three groups by designation.



## Event Handlers

Event handler is a script representing sequence of operators executed when the event appears. All scripts of this type can be found in the Script group property via the Property Grid.

Script	
Aggregates	(Collection)
GenerateScript	
GetDataScript	

Property	Object	Description
GenerateScript	All objects	Code executed before the object is generated
ManualBuildScript	Page	Code executing generation of the page content. If the script is set, standard rendering mechanism is unavailable
CommonScript	Document	Code of the main part of expressions. Here you can declare global variables, functions, procedures
GetDataScript	DataBand, CrossBand	Code of the expression that will be executed when the data is got from the data source

## Property Binding

Property binding is a script representing expression connected to the object property; its calculation result is set as a value for this property. Expressions can be set for the most object properties in the Property Grid on the Bindings tab. Expressions are calculated before the object rendering before the GenerateScript is called. The order of expressions calculation is not defined, that is why we don't recommend writing expressions which logic is based on the order of expressions calculation for different properties.

For example,

`textBox.Value = a + b`

`textBox.Style = (textBox.Value == 0) ? "Zero" : "Normal"`

There are no errors, but expression calculation result for the `textBox.Style` property can be invalid. It is necessary to set property

`textBox.Style = (a + b == 0) ? "Zero" : "Normal"`

## Properties-Expressions

Properties-Expressions are special properties getting value of the certain type. Properties-bindings are analogous to the property bindings, but they don't match any property. Result got by the expression is used but it is not stored in the property.

Properties-expressions are set in the Property Grid; you can use Script Editor to set them. There are properties representing collection of properties-expressions.

Properties-expressions:

`GroupBand.GroupExpression`

Calculation result sets order of grouping. Expression gets group key (if the key is equal, rows are in the same group). Calculated value is not set for the property but they are used in inner rendering algorithms.

(More information about grouping)

`GroupBand.SortExpression`

Calculation result sets order of sorting.

(More information about sorting grouped lists)

`DataBand.FilterExpression`

Calculation result sets rule of filtering data.

(More information about filtering lists)

`DataBand.Sort`

Collection of sorting conditions. Each sorting condition is an object with the Expression property that sets expression getting key by which the data is sorted.



(More information about lists sorting)

DataBand.Totals

Collection of aggregation functions expressions.

(More information about totals)

As these expressions are calculated for the inner logic of rendering time of calling and order of calculation is not set. You shouldn't create logic based on assumption about order or time of calculation. As a rule, such expressions are based on the source data value. Of course, it is possible to assume that scripts connected to initialization of the whole document and DocumentScripts script class will be executed at first.

### Document Object Model

It is possible to refer to all template objects and the document itself. You can refer to them by their name set in the Name property and using Document name.

It is possible to use all public methods and element types' properties (More information about types see in Class reference). It is possible because fields that correspond by type and name to the element added to the template are added to the DocumentScripts script class. These fields are initialized by the instances of the objects that will be used in the report rendering process. For example, you added TextBox with name "textBox1" to the template; TextBox textBox1 field of the private type will be added to the DocumentScripts. And before rendering starts it will be initialized by the certain instance of the TextBox object, and by the very instance that will be used in the rendering process. Now you can refer to this element in any script and set, for example, textBox1.Value = 1. This mechanism is applied to each element added in the visual designer. By adding an element in the visual designer you add description of the corresponding fields to the code.

(More information about Document Object Model)

### Methods and Properties

"DocumentScripts" class created during template design and writing scripts is a descendant from the special class: PerpetuumSoft.Reporting.Rendering.ReportScriptBase. This type has a lot of useful properties and methods that can be used in scripts. As written scripts are a part of code of the class inherited from Rendering.ReportScriptBase. (More information about type description is in Class reference).

### Work of Scripts during Report Rendering

TBD

### Access to Types, Basic Types and Their Methods

#### Access to the Base Class

Here are the most important properties and methods of the scripts base class.

Property or Method	Description
Now	Gets the date and time at which rendering began
PageCount	Gets the count of pages in the document
PageNumber	Gets the number of the current page
ColumnNumber	Gets the number of the current column
DataObjects	Gets the collection of associated data sources
Document	Gets the template of the currently rendered document
Engine	Gets the associated RenderEngine
Group	Gets current group when data is grouped
GetData	Returns data corresponding to specified data member
GetParameter	Returns parameter by specific name
GetTotal	Gets total value by the field. Depending on the place of reference, it can get total by group or by all entered data



GetRunningTotal	Gets running total by its name
-----------------	--------------------------------

Let's consider Engine property of the PerpetuumSoft.Reporting.Rendering.RenderEngine type, as it is a complicated object representing report generator engine.

Here are the most important properties and methods of RenderEngine:

Property or Method	Description
ColumnsCount	Gets columns count
LineNumber	Gets number of the current line
PageCount	Gets count of pages in the document
PageNumber	Gets number of the current page
Data on filling the current generated page:	
PageRenderArea	Gets and sets a rectangle defining the remaining free space available on the page
Caret	Gets and sets the current position on the page being rendered
UsedHeight	Gets the amount of used page height
UsedWidth	Gets the amount of used page width
FreeHeight	Gets the amount of remaining free page height
FreeWidth	Gets the amount of remaining free page width
Data sources and parameters:	
Objects	Gets the amount of remaining free page width
Parameters	Gets the amount of remaining free page width
Generated document and current page:	
ProductionDocument	Gets the final document
ProductionPage	Gets the current page in the final document
Data on the used document template and current used template page:	
TemplateDocument	Gets the document template
TemplatePage	Gets the document template

It is possible not only to learn current state of the rendering process, but also influence it by changing the corresponding values.

Method	Description
RenderEngine functions used to manage rendering process	
NewColumn	Creates new column
NewPage	Creates new page
CancelRender	Cancel rendering

### Access to Data

More often scripts are used to bind report elements to certain data. Setting of various properties is executed in the Property Binding by the source data. In order to get current value of the source field, GetData function is used. Path to data separated by the "." symbol is delegated as parameter.

Very often elements added inside DataBand display fields of one and the same source set in the DataBand.DataSource property. In this case it is more convenient to use DataBand indexer. It is worth to say that GetData, DataBand[] gets values of the current line. DataBand moves cursor from line to line during report generation. You can form such scripts by drag and drop fields from the data source tree in the Script Editor.

### Access to Rendering Engine

TBD

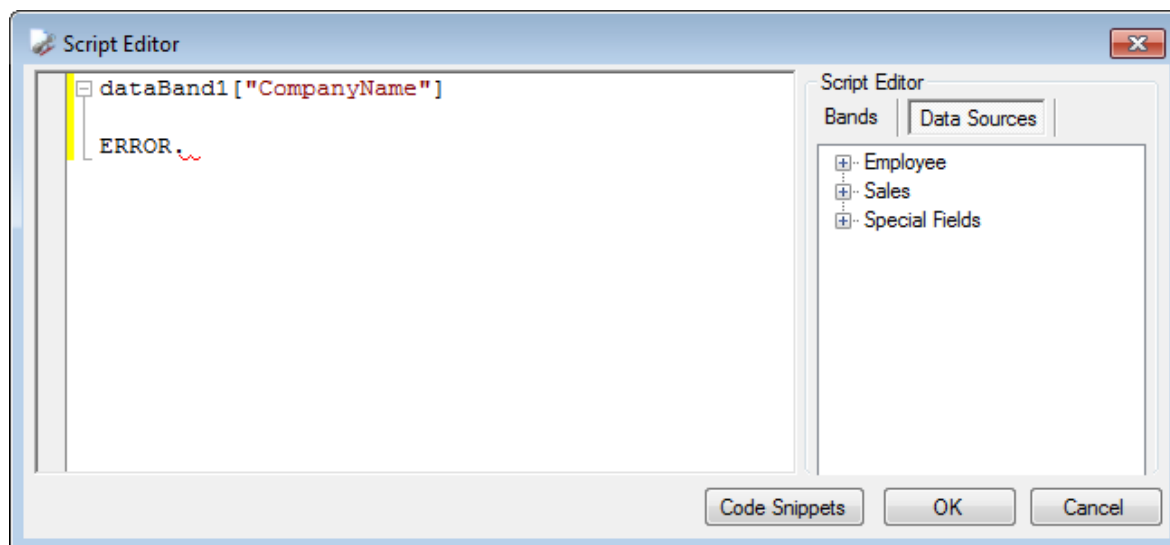
### Access to Other Assemblies

TBD

## Errors

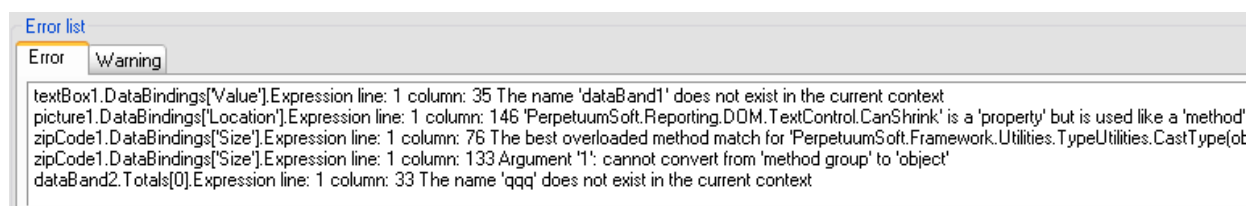
When you write scripts you can make various errors: syntax, semantic, errors in script logic.

Syntax errors are easy to notice as Script Editor features syntax highlighting and it underlines errors.



When you try to generate a document ok when you check scripts syntax and semantic errors will be listed in a specific error list.

(You can check scripts by the menu command "Report\Check Script" or by clicking "Check Script" button on the toolbar).



The list contains name of the band the error is contained in and its description. Double click on the list item and you will run Script Editor to edit script or expression.

To debug scripts, it is possible to display value of necessary variables in control points, for example, by using the following dialog: `System.Windows.Forms.MessageBox(var1.ToString())`.

We strongly recommend avoiding sophisticated logic in scripts; using only base functionality. If scripts are too sophisticated you need to review concept of the report template.

If you need to debug sophisticated logic of calculations, you can use a workaround. All such logic can be separated in a function(s). These functions are included in the application and scripts refer to the application for the required calculations. In the application, you can use debugger of the development environment you use. After you debugged the logic you can get function back to script code if you need to make template independent from the application.



## Typical Tasks and Solutions in Scripts

### Data Processing

TBD

### Managing Visibility

Sample displaying number of the current line; only odd lines are visible.

Report Template:

dataBand1:DataBand	
detail1:Detail	
<	dataBand1.LineN
end of dataBand1	

dataBand1.InstanceCount = 100 (property sets amount of times the band displays its content in the report)

textBox1.Value = dataBand1.LineNumber (output of the number of the current page)

detail1.Visibel (dynamic property, Binding) = "dataBand1.LineNumber % 2 == 0 ? false : true"

I.e. Detail object is visible if the current page is odd.

Report:

1
3
5
7
9
11
13

In case you manage visibility of the TextBox object (not Detail) there will be an empty line between odd lines – Detail object with invisible Textbox object.

### Managing Position

Sample demonstrating object position.

Report Template:

dataBand1:DataBand	
detail1:Detail	
<	data
end of dataBand1	

dataBand1.InstanceCount = 100 (property sets amount of times the band displays its content in the report)

textBox1.Value = dataBand1.LineNumber (output of the number of the current line)

textBox1.Location (dynamic property, Binding) =

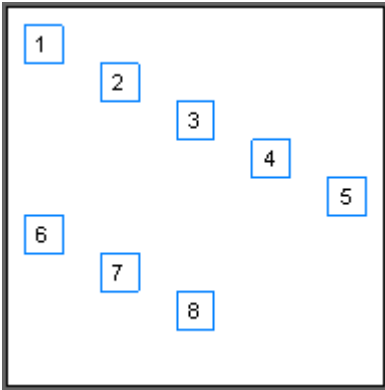
"PerpetuumSoft.Framework.Drawing.Vector(0.25f+1f\*((dataBand1.LineNumber - 1) % 5),0f).ConvertUnits(Unit.Centimeter,Unit.InternalUnit)"

Every five TextBox objects are displayed with the right shift by 1 cm.

ConvertUnits method will convert measures.



Report:



### Access to Application Logic

All objects from the application assemblies are available from scripts. Sample demonstrating the use of application function. Sample is located in the folder {SharpShooter Collection Installation Directory}\Samples\SharpShooter Reports\CSharp\HostingApplicationExample. GetString() function getting string "Hello from HostingApplicationExample.exe" is declared in the main form class.

```
public static string GetString()  
{  
    return "Hello from HostingApplicationExample.exe";  
}
```

In order to get access to the form class it is enough to add namespace the form is located in to the Document.Imports property.

Script	
CommonScript	
CompilerVersion	
CustomLanguageProvider	
GenerateScript	
Imports	HostingApplicationExample ...
ScriptLanguage	CSharp

GetString() function is called in the Value property of the textBox1 object.

```
<Form1.GetString()>
```



## Data Sources

### Data Source Classes

The following classes can be used as data sources for SharpShooter Reports:

ADO.NET objects – System.Data.DataSet, System.Data.DataView, System.Data.DataTable;

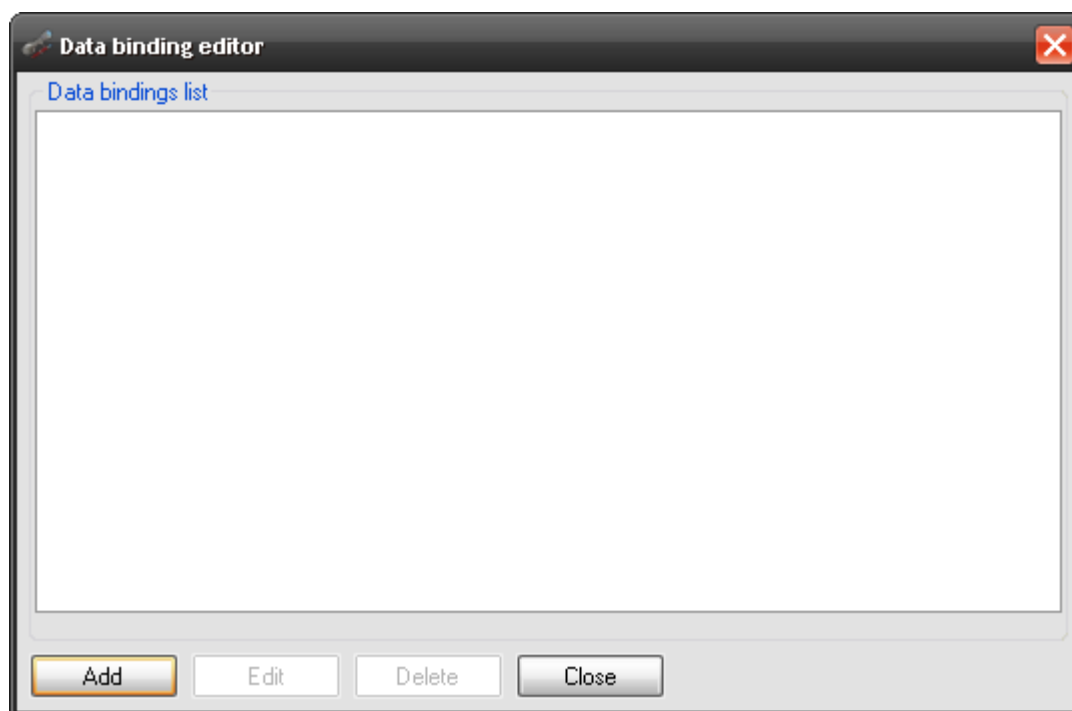
Custom classes (Business Objects) implementing System.ComponentModel.IListSource or System.Collections interface. IEnumerable (IEnumerable interface is implemented by many standard classes, for example, System.Array, System.Collections.ArrayList, System.Collections.CollectionBase and many others);

Any other custom classes (Business Objects) that are not included in the list above and all properties of this object will be available as data.

### Setting Data Source

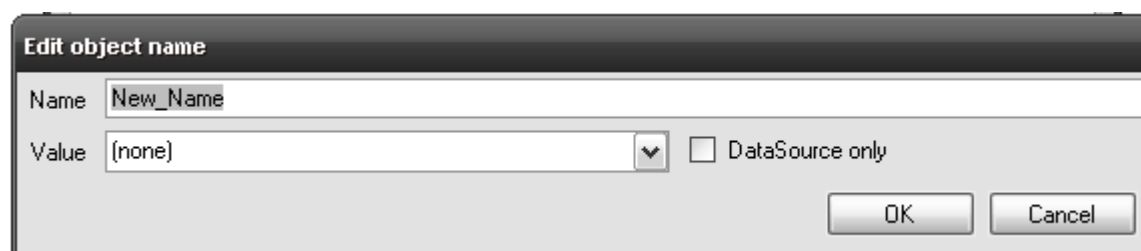
Report data sources are set in the DataSources properties of the ReportManager class. This property has a PerpetuumSoft.Reporting.Components.ObjectPointerCollection type. This class is a collection of objects bound to string keys.

To set report data sources via the property editor, select ReportGenerator object and in the Property grid, click the button of the DataSource property. There will appear a Data Binding form shown in the picture below



“Edit” button runs editor for the selected data source in the Data Binding List, “Delete” button deletes selected data source, “Close” button closes Data Binding Editor.

To add a new data source, click “Add” button and you will see form shown in the picture below.





Name field is used to enter name under which data source will be available in the report. Value field is used to select name of the object used in your application that is a data source itself. As a result the InitializeComponent() method will be added to the code shown below

(C#)

```
this.reportManager.DataSources =  
    new PerpetuumSoft.Reporting.Components.ObjectPointerCollection(  
        new string[] {"AccountsDataSet", "CustomersByCity"},  
        new object[] {this.accountsDataSet, this.customersByCity});
```

(VB)

```
Me.reportManager.DataSources =  
    New PerpetuumSoft.Reporting.Components.ObjectPointerCollection(  
        New String() {"AccountsDataSet", "CustomersByCity"},  
        New Object() {Me.accountsDataSet, Me.customersByCity })
```

To make data source display in the Value list, it should be descendant from the System.ComponentModel.Component class.

To add data source from code, you can use Add method of the ObjectPointerCollection class of the Item property:

(C#)

```
reportManager.DataSources.Add("DataSourceName", dataSource);
```

(VB)

```
reportManager.DataSources.Add("DataSourceName", dataSource)
```

or

(C#)

```
reportManager.DataSources["DataSourceName"] = dataSource;
```

(VB)

```
reportManager.DataSources("DataSourceName ") = dataSource
```

Add method accepts two parameters: data source name by which it will be available in the report and the data source itself. In the same way you can set name and data source when using Item property.

Report data sources can be loaded directly in the template by using report.

### *Using ADO.NET Objects*

Let's consider a sample of using ADO.NET as a data source. We have a database with two tables: "Authors" and "Books" storing authors' names and books they wrote. "Authors" table has two fields: primary key and author name, and "Books" table includes primary key, author key, book title and price. This sample is located in the ADODataSource folder.

### *Using Business Objects*

As it was mentioned above, in order to make custom data source represent data, it should implement System.Collections.IEnumerable or System.ComponentModel interface. IListSource (IEnumerable interface is implemented by many standard classes, for example, System.Array, System.Collections.ArrayList, System.Collections.CollectionBase and many others). IListSource interface defines two members: bool ContainsListCollection {get;} property, it doesn't influence work of SharpShooter Reports; and GetList() method getting reference to System.Collections.IList interface that in its turn implements IEnumerable interface.

In any case, work with custom data is executed via the IEnumerable interface that defines a single GetEnumerator() method getting reference to the System.Collections.IEnumerator interface. This interface enables you to logically view all elements of the collection. IEnumerator interface defines object Current {get;} property representing current object in the collection and



two methods: `bool MoveNext()` and `void Reset().MoveNext()` method executes move to the next collection element and gets true if successful, and false, if the current element is last in the collection. `Reset()` method moves numerator to initial state, i.e. current position is a position before the first collection element.

All properties will be available as data fields of the objects stored in custom data arrays. In its turn, if the property has a type implementing `IListSource` or `IEnumerable` interface, it is considered as custom data array. It allows the creation of complicated reports with master-detail relations using custom data sources.

If data source implements neither `IListSource` nor `IEnumerable` interface, then all its properties will be available as data. So, we can implement one record from the table.

Now we can create custom data sources for the authors and books data sources considered in the section "Using ADO.NET objects".

Create the following class to store information on a single book

```
public class Book
{
    public Book()
    {
    }
    private string name = string.Empty;
    public string Name
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
    private decimal price;
    public decimal Price
    {
        get
        {
            return price;
        }
        set
        {
            price = value;
        }
    }
}
```

This class contains two properties that will be available during report creation.

Now add the class to the `BookCollection` to store instances of the `Book` class implementing `IEnumerable` interface. We will store data in the `ArrayList` class instance. We will also implement two methods: `Add()` – to add element to the collection and `GetEnumerator`, and implement `IEnumerable` interface in this way. Code of this class shown below:

```
public class BookCollection : IEnumerable
{
    private ArrayList list = new ArrayList();

    public BookCollection()
    {
```



```
}  
public void Add(Book b)  
{  
    list.Add(b);  
}  
public IEnumerator GetEnumerator()  
{  
    return list.GetEnumerator();  
}  
}
```

Further you need to create class to store information on a single author. As author can have multiple books, the property storing the books should be a collection. So, we realize master-detail relation in custom data sources.

```
public class Author  
{  
    public Author()  
    {  
    }  
    private string name = string.Empty;  
    public string Name  
    {  
        get  
        {  
            return name;  
        }  
        set  
        {  
            name = value;  
        }  
    }  
    private BookCollection books = new BookCollection();  
    public BookCollection Books  
    {  
        get  
        {  
            return books;  
        }  
    }  
}
```

You need to create a collection to store authors. For example, this class can be a descendant from `System.Collections.CollectionBase` that is a base class for strongly typed collections and that implements `IEnumerable` interface.

```
public class AuthorCollection : CollectionBase  
{  
    public Author Add(Author value)  
    {  
        base.List.Add(value as object);  
        return value;  
    }  
    public void Remove(Author value)  
    {  
        base.List.Remove(value as object);  
    }  
    public void Insert(int index, Author value)  
    {  
        base.List.Insert(index, value as object);  
    }  
    public Author this[int index]  
    {  
    }  
}
```



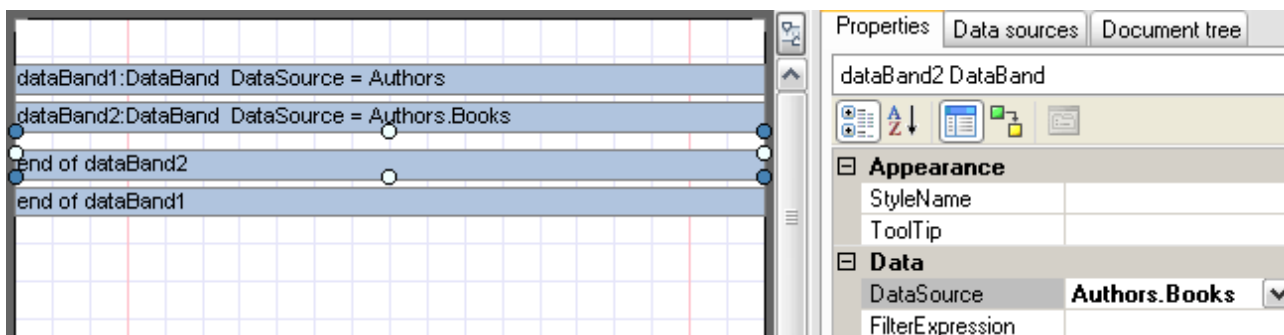
```
get
{
    return (base.List[index] as Author );
}
set
{
    base.List[index] = value;
}
}
}
```

In this sample, BookCollection class implements IEnumerable interface only to show that there is such ability. Of course, collections can be stored in arrays and in ArrayList class instances, but it is better to use strongly typed collections (descendants of CollectionBase).

Finally, you need to write the method to fill the data source and add it to the DataSources collection of the ReportManager.

```
private void Init()
{
    Author a = new Author();
    a.Name = "Kent Beck";
    Book b = new Book();
    b.Name = "Extreme Programming";
    b.Price = 7.45M;
    a.Books.Add(b);
    b = new Book();
    b.Name = "Extreme Programming Explained";
    b.Price = 11.33M;
    a.Books.Add(b);
    authorDS.Add(a);
    a = new Author();
    a.Name = "Craig Larman";
    b = new Book();
    b.Name = "Applying UML and Patterns";
    b.Price = 15.80m;
    a.Books.Add(b);
    authorDS.Add(a);
    reportManager1.DataSources.Add("Authors", authorDS);
}
```

To create Master-Detail report template, in Report Designer put one DataBand inside other DataBand. Set dataBand1.DataSource property to "Authors" and dataBand2.DataSource property to "Authors.Books".



To get access to data source fields set `textBox.Value = <DataBand name>[<Field name>]`.



dataBand1:DataBand DataSource = Authors	
detail2:Detail	
<	<dataBand1["Name"]>
dataBand2:DataBand DataSource = Authors.Books	
detail1:Detail	
<	<dataBand2["Name"]   <dataBand2["Price"]>
end of dataBand2	
end of dataBand1	

This sample is located under the {SharpShooterCollection Installation Directory}\Samples\SharpShooter Reports\CSharp\UserDataSource folder.

### Non-Standard Ways of Using Business Objects

In case you need to change standard mechanism of working with custom data source so that some dynamically calculated properties (not object properties) are available as fields, you need to implement ICustomPropertyDescriptor interface for the object representing a single record in the data source and ITypedList interface for the collection storing these objects.

Let's consider this possibility by example. Suppose that you develop application for the company selling parts all over the world. So, in different countries, you will need to output price in different currencies calculating them from the base one. This sample is located in the CustomPropertyDescriptorExample folder.

Currency class is used to store information on all possible currencies. Class contains two properties that set name and exchange rate. Static array SystemCurrencies is declared in the class; this array is used to store all currencies used in the system. In the sample, array has definite value, but in real applications, nothing prevents you from loading this array from the database.

To store a single record about the part, Path class is used; this class implements ICustomPropertyDescriptor interface. Below you can see code of the static methods forming collection of dynamic properties of this class.

```
public static PropertyDescriptorCollection GetPartProperties()
{
    PropertyDescriptorCollection props = new PropertyDescriptorCollection(null);
    foreach(Currency c in Currency.SystemCurrencies)
    {
        props.Add(new CurrencyPropertyDescriptor(c));
    }
    props.Add(TypeDescriptor.CreateProperty(typeof(Part), "Name", typeof(string)));
    return props;
}
```

This very method is called by the interface methods

```
ICustomPropertyDescriptor.GetProperties(Attribute[] attributes)
ICustomPropertyDescriptor.GetProperties()
```

In order to get collection of dynamic properties. As you can see from code, at the beginning of the cycle CurrencyPropertyDescriptor class instances for every currency type are created from the Currency.SystemCurrencies array and then part name is added.

CurrencyPropertyDescriptor class is inherited from the PropertyDescriptor class that defines abstraction for the class property definition. Pay attention to implementation of the PropertyType and ComponentType properties of this class that gets type of the property and type of the class implementing this property as well as to two methods: GetValue and SetValue getting and setting property value correspondingly.

And finally, let's consider PartCollection class used to store parts collection. This class implements ITypedList interface. This interface contains two methods: GetListName getting list name and



GetItemProperties getting array of PropertyDescriptor objects defining dynamic properties of the list objects. This method just gets result of the work of static Part.GetPartProperties() method.

So, you will need to create report template displaying price in necessary currency for every country.

### *Using XML File as Data Source*

To use XML document as data source you need to perform the following steps:

1. Create PerpetuumSoft.Reporting.Data.XmlDataSource object.
2. Set path to XML file in the DataFile property.
3. Set initial path (expression in XPath language) applied to XML data in the XPath property.
4. Set created object as data source for ReportManager.

You can do it by using ReportManager.DataSources.Add("DataSourceName", "DataSource") method.

You can use this data source in the template as any other data source.

Data source will contain as many records as there are elements found in the initial XML document by the set initial path (XPath property).

To get data for the report template use GetData("SourcePath") method.

Here, SourcePath is a string specifying data source name and then coma separated path to elements in accordance with XML document structure.

If SourcePath sets path to XML tag attribute, then GetData gets string value of this attribute. To get text contained inside XML tag, in the SourcePath property, set path to the tag and then coma separated "GetText()".

Let's review a sample:

XML document containing data on books is used as data source.





```
- <bookstore xmlns:bk="urn:samples">
- <book genre="novel" publicationdate="1997" bk:ISBN="1-861001-57-8">
  <title>Pride And Prejudice</title>
  - <author>
    <first-name>Jane</first-name>
    <last-name>Austen</last-name>
  </author>
  <price>24.95</price>
</book>
- <book genre="novel" publicationdate="1992" bk:ISBN="1-861002-30-1">
  <title>The Handmaid's Tale</title>
  - <author>
    <first-name>Margaret</first-name>
    <last-name>Atwood</last-name>
  </author>
  <price>29.95</price>
</book>
- <book genre="novel" publicationdate="1991" bk:ISBN="1-861001-57-6">
  <title>Emma</title>
  - <author>
    <first-name>Jane</first-name>
    <last-name>Austen</last-name>
  </author>
  <price>19.99</price>
</book>
</bookstore>
```

It is necessary to create report displaying data on all books. Here are the required steps:

1. Create XmlDataSource data source, set this XML file in the DataFile property, and "book"(XPath='bookstore/book') path in the XPath property.
2. Add created data source to the data source collection under "XmlSource" name by using the ReportManager.DataSources.Add method.
3. Create report template containing DataBand, set data source name for it DataBand.DataSource = "XmlSource".
4. To display data on every book ("book"), add Detail inside DataBand.
5. Add TextBox elements inside Detail to view various data on book. For example, to view book title ("title" element), set TextBox.Value(property binding) = 'GetData("XmlSource.title.GetText())'. To view data on author name, set 'GetData("XmSource.author.first-name.GetText())'. To view data from XML attribute, for example, from 'publicationdate', set 'GetData("XmSource. publicationdate ")'



Report template should look as follows:

dataBand1:DataBand DataSource = XmlSource				
header2:Header				
Title	First name	Last name	Date	Price
detail1:Detail				
<GetData ("XmlSource.title.GetText())"	<GetData ("XmlSource.author.first-na	<GetData ("XmlSource.author.last-na	<GetData ("XmlSourc	<GetData ("XmlSourc
end of dataBand1				

As a result you get the following document:

Title	First name	Last name	Date	Price
Pride And Prejudice	Jane	Austen	1997	\$24.95
The Handmaid's Tale	Margaret	Atwood	1992	\$29.95
Emma	Jane	Austen	1991	\$19.99

In some cases it is necessary to set XML document as data source directly in the template designer. There is such a possibility (review section "Direct access to data base").

### Relational DBMS

TBD

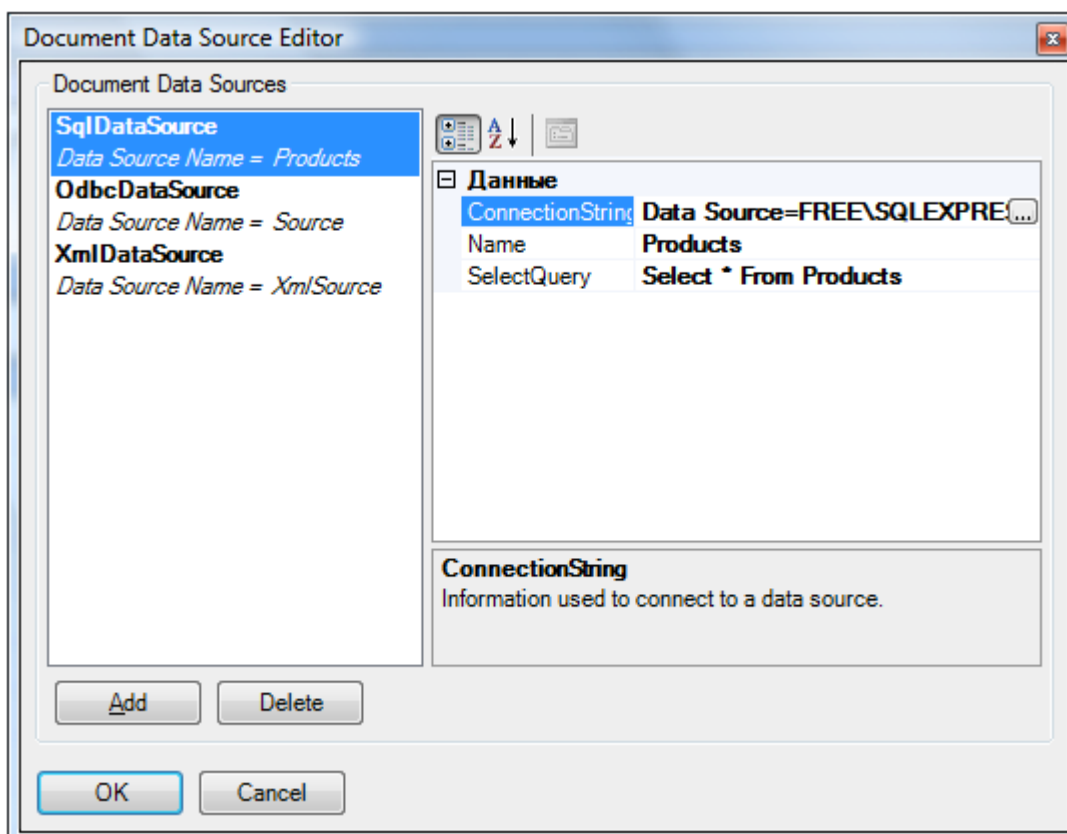
### Direct Access to Data Base

SharpShooter Reports report generator realizes the following concept: data for the report should be provided by the application. It allows the use of any data sources and makes rules of providing data (report template) independent form the certain data source.

But sometimes there is a requirement to strongly bind report template to certain data source, or to provide end user the ability not only to create/edit templates, but add custom data sources.

That is why report template features a collection of built-in data sources – data sourceset directly for this template (Document.DataSources property). Report designer allows adding and editing such data sources.

Built-in Document Data Source Editor



Editor is called when you set Document.DataSources property or click "Data Sources" button on the toolbar.

You can set or edit the following data source types in the editor:

- 1) OdbcDataSource to get data from different DBMS via the Open Database Connectivity (ODBC) interface.
- 2) SqlDataSource to get data from MS SQL.
- 3) XmlDataSource to get document data.

The use of data sources built-in the document template doesn't differ from the use of data sources connected from the application. Built-in data sources and their structure (if it is possible to get data from this data source during template design) will be available in the template designer in the data sources tree.

To get data in the template directly from data base it is possible to use OdbcDataSource or SqlDataSource data source. Their designation and properties are similar; they differ only by the way of connection to DBSM. If you use MS SQL, it is recommended to use SqlDataSource.

To set OdbcDataSource or SQL SqlDataSource data source you need to set:

1. Name property – sets data source name used to refer to data source from a template.
2. ConnectionString property – sets connection string to data source.
3. SelectQuery property – sets SQL query to get data.

To get data in the template directly from XML documents, it is necessary to add XmlDataSource data source, set its properties:

1. Name property - sets data source name used to refer to data source from a template.



2. DataFile property – sets path to XML document
3. XPathproperty – sets initial path (expression on XPath language) applied to XML data.

More information about use of XmlDataSource see in the section “Using XML file as data source”

### *Report without Data*

Reports can be created without setting data source. DataBand object contains InstanceCount property that sets count of times the object will output its content in the report. Sample of the report without data is located in the WithoutDataExample folder. This sample outputs multiplication table. Report template contains two DataBands. DataBand2 is located inside dataBand1. InstanceCount property of both elements is set to 10. Value property of the textBox1 objects has the following value:

```
dataBand1.LineNumber.ToString() + " * " + dataBand2.LineNumber.ToString() + " = " +  
(dataBand1.LineNumber * dataBand2.LineNumber).ToString()
```

LineNumber property of the DataBand shows the number of the current line. So, we display multiplication table for 1, then for 2 and so on up to 10.

### *Non-Standard Data Sources*

TBD



### Export

SharpShooter Reports allows saving final documents to popular formats. Export filters are used for this purpose. They have overloaded Export function:

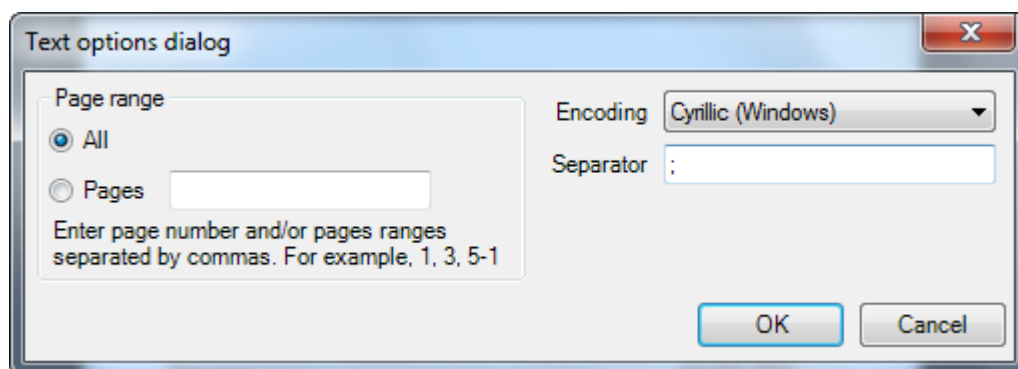
```
public void Export(Document document, string fileName )
public virtual void Export(Document document, string fileName, bool showDialog),
```

where document is a final report, filename is name of a file in which exported document will be saved, showDialog sets visibility of the export settings window. If the first method is used, dialog window of export filter settings will be shown. Export to formats which filters are located in the assembly PerpetuumSoft.Reporting, is always available from Report Viewer. In order to use other export filters it is necessary to connect to the corresponding assembly and create at least one instance of the class of this filter. The simplest way to do it is to add necessary filter to the form. Besides, to manipulate available export filters, developer has access to static method PerpetuumSoft.Reporting.Export.RegisterExportFilter(ExportFilterFactory factory) and static collection PerpetuumSoft.Reporting.Export.ExportFilters.

All formats have standard settings: Page range sets range of exported report pages; Encoding sets text coding; Resolution sets images resolution.

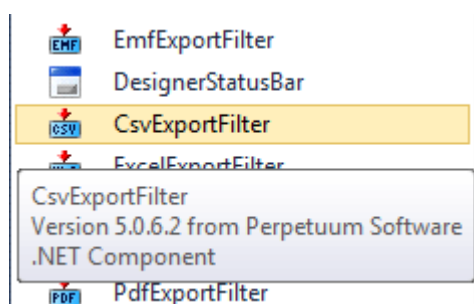
### CSV Export

Only text information is exported to CSV. Content of the TextBox and AdvancedText elements is exported to CSV. Every line of a file corresponds to a line of a table. Values of separate columns are separated by separation symbol that can be set in the Separator field.



CSV export filter is located in the PerpetuumSoft.Reporting.Export.Text assembly.

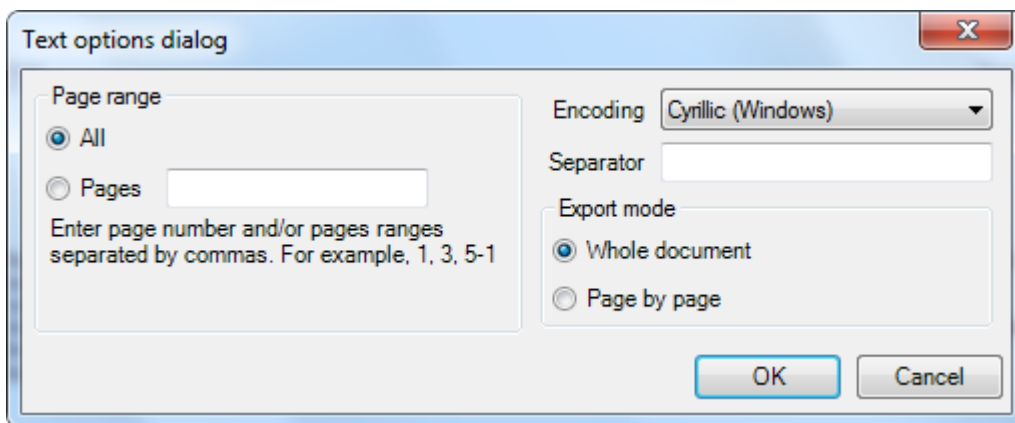
It is possible to add CSV Export Filter to the project in Visual Studio using Toolbox:



### TXT Export

Content of the TextBox and AdvancedText elements is exported to TXT. In the Export Mode settings, you can select generation mode. "Page by Page" mode adds symbols of page break. If you select "Whole Document" mode there will be no such symbols.

TXT Options Dialog:

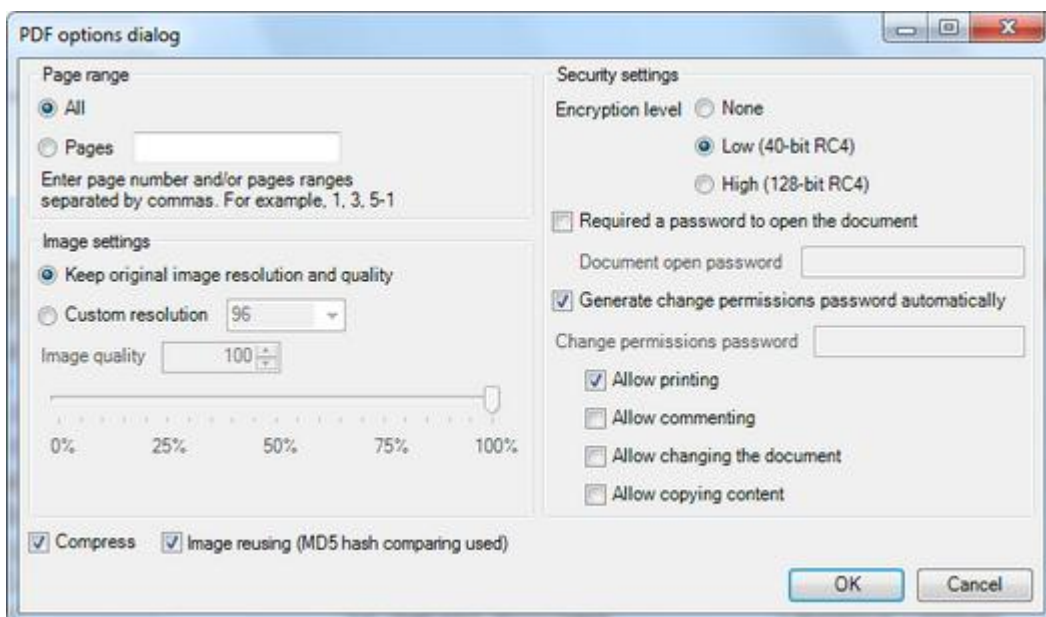


TXT export is located in the PerpetuumSoft.Reporting.Export.Text assembly.

### PDF Export

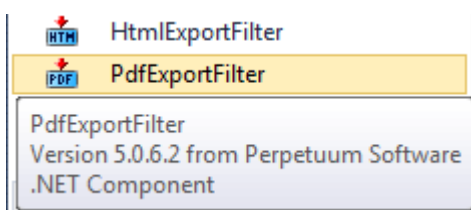
TextBox and AdvancedText elements are exported to PDF, styles are saved. Picture, ZipCode, BarCode, RichText, WinFormsControl, Chart, Widget elements are exported as pictures. ConicalFill and HatchFill are exported as SolidFill. It is possible to set security settings for the PDF document and quality settings for images. The "Image Reusing" option allows the optimizing final document size if the exported report contains repeated pictures.

PDF Options Dialog:



PDF export filters are located in the PerpetuumSoft.Reporting.Export.Pdf and PerpetuumSoft.Writers.Pdf assemblies.

It is possible to add PDF Export Filter to the project in Visual Studio using Toolbox:

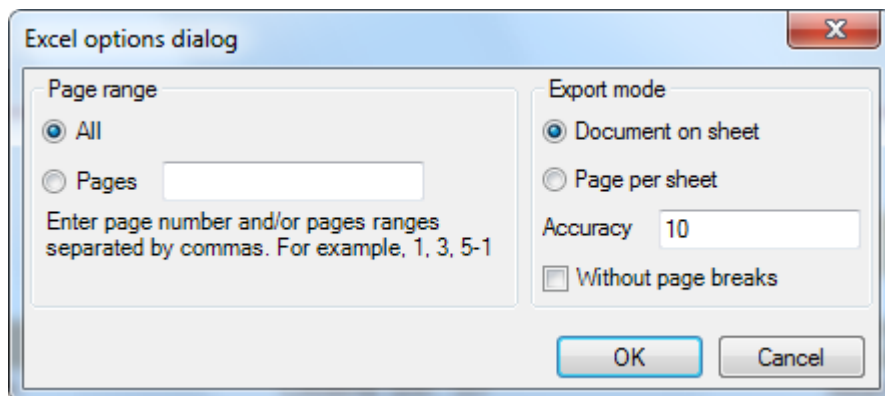




### Excel Export

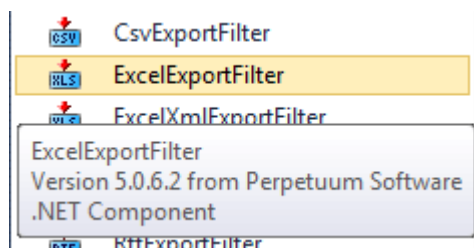
Text information from the TextBox and AdvancedText elements as well as Shapes are exported to Excel. All types of Fill and TextFill are exported as SolidFill. In the Export Mode settings, you can select way of forming pages – to place all report pages on a single page or place report page on separate document pages. The “Without page breaks” option allows the exporting all data rows without empty lines.

Excel Options Dialog:



Excel export filters are located in the PerpetuumSoft.Reporting.Export.Excel and PerpetuumSoft.Writers.Excel assemblies.

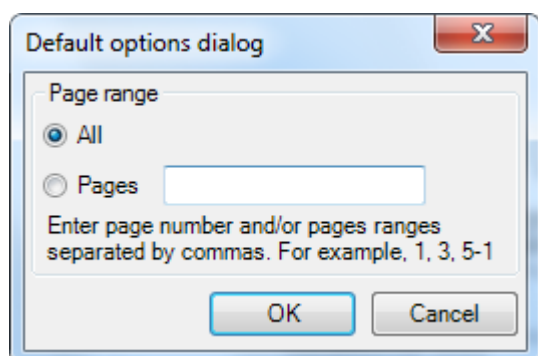
It is possible to add Excel Export Filter to the project in Visual Studio using Toolbox:



### Excel (XML) Export

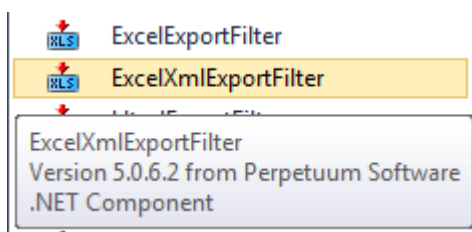
Content of the TextBox and AdvancedText elements is exported to Excel (XML), styles are saved, and all types of Fill and TextFill are exported as SolidFill.

Default Options Dialog:



Excel XML export filters are located in the PerpetuumSoft.Reporting.Export.ExcelXML and PerpetuumSoft.Writers.Excel assemblies.

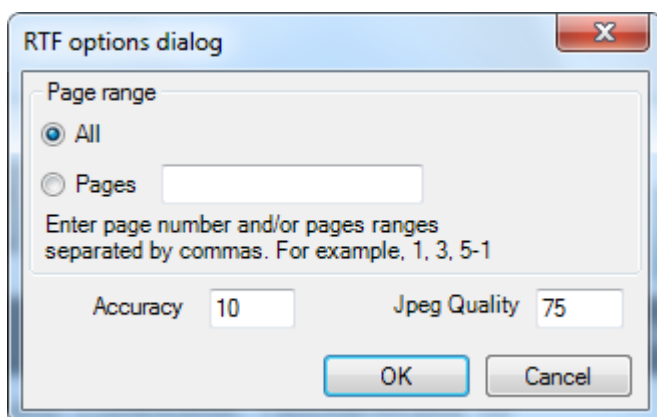
It is possible to add Excel XML export filter to the project in Visual Studio using Toolbox:



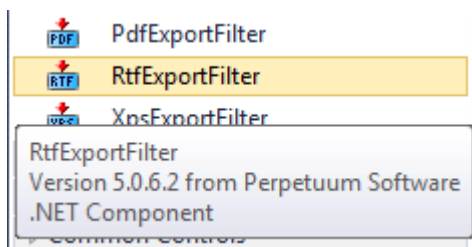
### RTF Export

Content of the TextBox and AdvancedText elements is exported to RTF, styles are saved. Picture, Shape, ZipCode, BarCode, RichText, WinFormsControl, Chart, Widget elements are exported as pictures. All types of Fill and TextFill are exported as SolidFill.

RTF Options Dialog:



It is possible to add Export Filter to the project in Visual Studio using Toolbox:

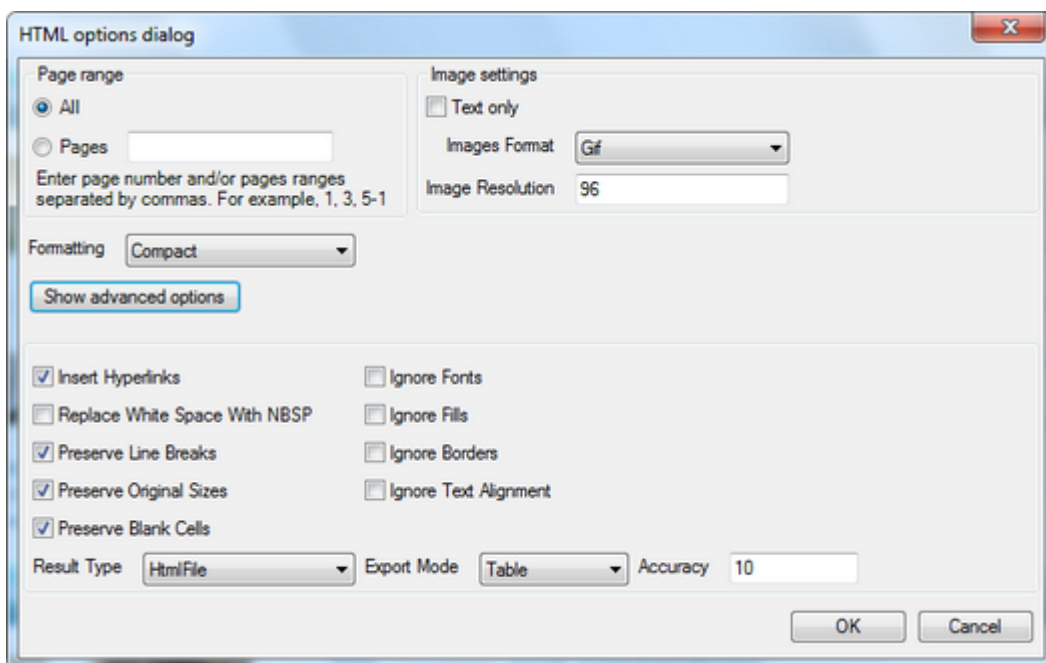


### HTML Export

Content of the TextBox and AdvancedText elements is exported to HTML, styles are saved. Picture, ZipCode, BarCode, RichText, WinFormsControl, Chart, Widget elements are exported as pictures. Shape element is not exported. To save only text information it is necessary to check Text Only in the export settings. All types of Fill and TextFill are exported as SolidFill. Advanced settings allow you to set additional export parameters.

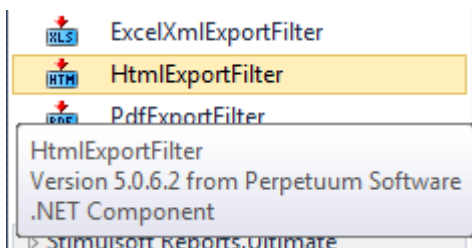
HTML Options Dialog:





HTML export filter is located in the PerpetuumSoft.Reporting.Export.Html assembly.

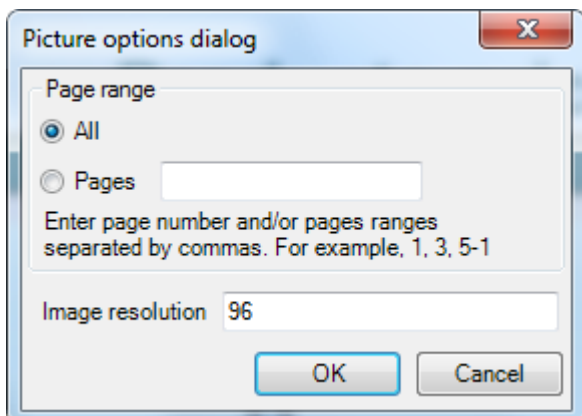
It is possible to add HTML Export Filter to the project in Visual Studio using Toolbox:



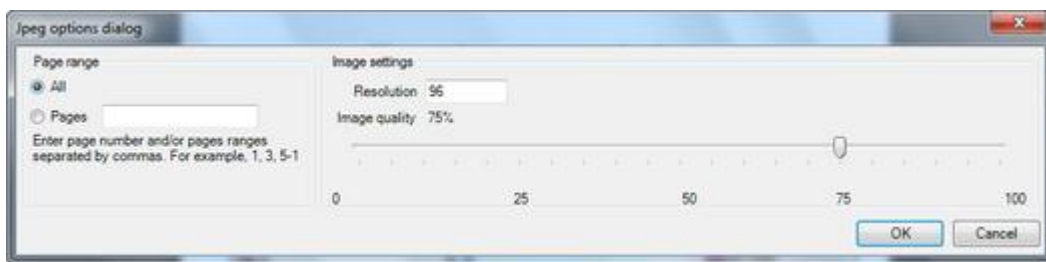
### ***GIF, JPG, BMP, EMF, TIFF, PNG Export***

All report's elements are included in the document. Every page is exported as separate file (except TIFF export). It is possible to set image resolution for bitmapped and vector (EMF) images and also image quality for JPG export.

Picture Options Dialog:

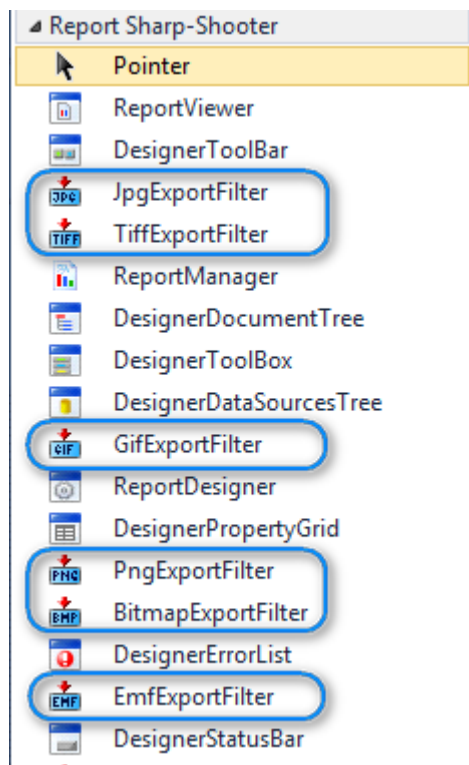


Jpeg Options Dialog:



Export filter for generation of bitmapped and vector (EMF) images is located in the PerpetuumSoft.Reporting assembly.

It is possible to add Export Filters to the project in Visual Studio using Toolbox:





## Projects

### Win Forms

#### Using Report Viewer

Report Viewer is a separate component having a set of properties that allow setting its appearance. These properties can switch on/off visibility of the status bar, toolbar and separate buttons and groups of buttons on the toolbar. Below is the list of these properties.

Property	Description
ShowContent	Show or not the "Show content" button and document table of contents
ShowDesigner	Show or not the "Edit report" button
ShowExport	Show or not the "Export document" button
ShowFind	Show or not the "Find text" button
ShowNavigator	Show or not the "Move backwards" and "Move forward" buttons
ShowOpen	Show or not the "Open document"
ShowPageNavigator	Show or not buttons: "Go to first page", "Go to previous page", "Go to next page", "Go to last page"
ShowPrint	Show or not the "Print document" button
ShowRefresh	Show or not the "Refresh report" button
ShowSave	Show or not the "Save document" button
ShowScale	Show or not buttons: "Pan mode", "Dynamic zoom mode", "Zoom in mode", "Zoom out mode", "Zoom to rectangle", "Fit to whole page", "Fit to page width", "Actual size"
ShowStatusBar	Show or not status bar
ShowStatusBarGrip	Show or not Grip on the status bar
ShowToolBar	Show or not toolbar

Report Viewer has Actions property that provides access to all actions and allows you to define it. Here is the list of all actions.

Action	Description
Print	Called when "Print document" button is clicked
Load	Called when "Open document" button is clicked
Save	Called when "Save document" button is clicked
Export	Called when "Export document" button is clicked
Pan	Called when "Pan mode" button is clicked
DynamicZoom	Called when "Dynamic zoom mode" button is clicked
ZoomIn	Called when "Zoom in mode" button is clicked
ZoomOut	Called when "Zoom out mode" button is clicked
ZoomRectangle	Called when "Zoom to rectangle" button is clicked
Content	Called when "Show Content" button is clicked
WholePage	Called when "Fit to whole page" button is clicked
PageWidth	Called when "Fit to page width" button is clicked
ActualSize	Called when "Actual size" button is clicked
Zoom25	Sets zoom 25%
Zoom50	Sets zoom 50%
Zoom75	Sets zoom 75%
Zoom100	Sets zoom 100%
Zoom125	Sets zoom 125%
Zoom150	Sets zoom 150%
Zoom200	Sets zoom 200%
Zoom300	Sets zoom 300%
Zoom500	Sets zoom 500%
Forward	Called when "Move forward" button is clicked
Backwards	Called when "Move backwards" button is clicked
CustomZoom	Called when the following hotkey is pressed Ctrl+Z or "Custom



	zoom" contextual menu item is selected for the Zoom block on the status bar
FirstPage	Called when "Go to first page" button is clicked
PrevPage	Called when "Go to previous page" button is clicked
NextPage	Called when "Go to next page" button is clicked
LastPage	Called when "Go to last page" button is clicked
GotoPage	Called when you click mouse on the status bar that displays information on the current page and total count of pages.
EditReport	Called when "Edit report" button is clicked
RefreshReport	Called when "Refresh report" button is clicked
SinglePage	Called when "Single page mode" button is clicked
ContinuedPage	Called when "Continued page mode" button is clicked

In order to redefine the action set by default, it is necessary to create Executing event handler. If the Handled property of the ExecutingEventArgs object delegated to the event handler is set to true, then standard action defined in the Execute event won't be called. For example, to redefine Export action:

```
reportViewer1.Actions["Export"].Executing += new
Action.ExecutingEventHandler(Export);
and create event handler:
private void Export(object sender, ExecutingEventArgs e)
{
    // Do something

    e.Handled = true;
}
```

#### Without Report Viewer

TBD

### Web Forms

#### Peculiarities of the Working Web

When you use SharpShooter Reports in WebForms applications, reports are always generated on the server side and then sent to the client. In this case you can send the report to the client in various formats: html, pdf etc. You can send the entire report or only the requested page at once. Below we will see various examples of using SharpShooter Reports in web applications.

#### Using SharpShooterWebViewer

SharpShooter Reports has a special web control called PerpetuumSoft.Reporting.Web.SharpShooterWebViewer for viewing reports from ASP.NET applications. This component has the property for customizing the ViewMode mode of viewing reports that can process the following values from the PerpetuumSoft.Reporting.Web.ViewMode enumeration:

- HtmlSinglePage – view a single page;
- HtmlWholeReport – view the entire report;
- WindowsForms – view the report in PerpetuumSoft.Reporting.View.ReportViewer (the component for viewing reports in Windows Forms applications).

The source of the SharpShooterWebViewer document is specified in the Source property of the ReportBase type from which the ReportSlot component is inherited. SharpShooterWebViewer also allows you to view the entire report in the pdf format. SharpShooterWebViewer caches all viewed documents; the time for reports to be stored in the cache is specified in the CacheTimeout property. Using properties from the Page category, you can customize the way the line allowing navigating through pages in the report will look like, NextText defines text for the link to the next



page, PagePosition defines the position of the navigation line, etc. The ImageFormat property defines the format in which pictures from the report are sent.

The WebDemo example demonstrates the use of SharpShooterWebViewer.

### Use of the Product without SharpShooterWebViewer

WebPublish sample demonstrates how to view reports on a Web page without using SharpShooterWebViewer. Report title and format of the report is set on the main page. After you click "Show" button, the query in the Click event handler is sent to the ReportPage.

```
Response.Redirect("ReportPage.aspx?format="+DropDownList1.SelectedItem.Value.ToString()+ "&title="+TextBox1.Text)
```

Event handler of the Load event of ReportPage page executes export of the report export to the necessary format and sends it to the client.

### Using HttpHandler

The other way of viewing report without SharpShooterWebViewer is to implement System.Web.IHttpHandler interface that allows the creation of a custom handler of Http events and set it to a definite extension, for example, \*.rst. This event handler will get document generated by the template set in the query. IHttpHandler is used in the HttpHandler sample. Query handling is executed in the ProcessRequest method of the ReportHttpHandler class. Query handler just exports report in the requested format and writes it to the reply thread. In order to use this handler, it is necessary to include the following information in the Web.config.

```
<configuration>
  <system.web>
    <httpHandlers>
      <add verb="*" path="*.rst" type="HttpHandler.ReportHttpHandler,HttpHandler"/>
    </httpHandlers>
  </system.web>
</configuration>
```

The verb attribute defines the list of HTTP commands for which this handler will be called, the path attribute defines the URL or the URL mask to call this handler for, and the type attribute defines the handler class and the assembly that contains it.

In our case, we should also click the "Settings" button on the "Home directory" tab in the web server settings and assign the .rst extension to {disk}:\{Microsoft.NET Framework path}\aspnet\_isapi.dll. on the "Assignment" tab.

Property	Description
ShowContent	Show/Do not show the Show content button
ShowDesigner	Show/Do not show the Edit report button
ShowExport	Show/Do not show the Export document button
ShowFind	Show/Do not show the Find text button
ShowNavigator	Show/Do not show the Move backwards and Move forward buttons
ShowOpen	Show/Do not show the Open document button
ShowPageNavigator	Show/Do not show the Go to first page, Go to previous page, Go to next page, Go to last page buttons
ShowPrint	Show/Do not show the Print document button
ShowRefresh	Show/Do not show the Refresh report button
ShowSave	Show/Do not show the Save document button
ShowScale	Show/Do not show the Pan mode, Dynamic zoom mode, Zoom in mode, Zoom out mode, Zoom to rectangle, Fit to whole page, Fit to page width, Actual size buttons
ShowStatusBar	Show/Do not show the status bar
ShowStatusBarGrip	Show/Do not show Grip on the status
ShowToolBar	Show/Do not show the toolbar



## Using Advanced Product Features

### Custom Report Designer

If you are not satisfied with the standard Report Designer interface, you can easily design custom one, as all its elements (toolbar, status line, errors list, etc.) are available as separate components. Below is a complete list of these components:

DesignerDataSourceTree is a component displaying data sources. In standard Report designer, it is located on the Data Sources tab of the Tool Window.

DesignerErrorList is a component displaying list of errors in scripts. In standard Report designer, it appears when you call check script function of form report in the lower part of the form, if the script contains errors.

DesignerPropertyGrid is a component displaying a list of properties of the selected element. In standard Report designer, it is located on the Properties tab of the Tool Window.

DesignerStatusBar is a Report Designer status bar.

DesignerToolBar is a Report Designer toolbar.

DesignerToolBox is a toolbox. In standard Report designer, it is located on the left of the form.

ReportDesigner is a main component displaying edited report template. Report Designer connects all other components. You just need to set their Designer property so that it specifies the corresponding ReportDesigner.

Sample of the custom Report Designer is located in the CustomDesignerViewer folder. There is a "Custom Designer" button on the form to call custom Report Designer. When you click this button `button1_Click` event handler is executed. The first line sets Report Designer form:

```
CustomDesignerForm f = new CustomDesignerForm();
```

This form contains declared Designer property of the ReportDesigner type used to set data sources for report, subreport and edited report template.

```
f.Designer.DataSourceManager = reportManager1.DataSources;  
f.Designer.SubReportResolver = reportManager1;  
f.Designer.Document = reportSlot1.LoadTemplate();  
Then we show the form:  
f.ShowDialog();
```

Let's redefine handler for the click on the "Preview Report" button so that the final report displays in the custom Report Viewer. Use Actions property of the ReportDesigner component. This property is a collection of actions. The action we need is Preview. When you call the action Executing event is called, then - Execute event. If the Executing event handler gets through parameters that the event is handled, then the Execute event won't be called. When you call Execute event, standard handler for this event is called. So, we need to create custom Executing event handler that calls our Report Viewer and sets flag informing that the event is handled. This event handler is shown below.

```
private void Preview(object sender, ExecutingEventArgs e)  
{  
    if (reportDesigner1.Document != null)  
    {  
        ReportManager manager = new ReportManager();  
        InlineReportSlot slot = new InlineReportSlot();  
        manager.Reports.Add(slot);  
        manager.DataSources = this.DataSources;  
        slot.SaveReport(this.Document);  
        manager.OwnerForm = ParentForm;
```



```
manager.ResolveSubReport += new
ResolveSubReportEventHandler(manager_ResolveSubReport);
slot.RenderCompleted += new EventHandler(slot_RenderCompleted);
slot.HyperlinkClick += new HyperlinkEventHandler(slot_HyperlinkClick);
slot.GetReportParameter += new
GetReportParameterEventHandler(slot_GetReportParameter);
slot.Prepare();
e.Handled = true;
}
}
```

```
private void slot_RenderCompleted(object sender, EventArgs e)
{
    ReportSlot slot = sender as ReportSlot;
    using (ViewerForm f = new ViewerForm(slot))
    {
        f.WindowState = FormWindowState.Maximized;
        f.ShowDialog(this);
    }
}
```

Set this procedure as the Executing event handler for the Preview action  
reportDesigner1.Actions["Preview"].Executing +=  
new Action.ExecutingEventHandler(Preview);

## Custom Report Viewer

Sometimes it is necessary to use standard interface of the standard Report viewer component, for example, when it doesn't fit application style or you need to change toolbar.

Let's create Report Viewer that doesn't contain toolbar (you just need to set ReportViewer.ShowToolBar property to false) and add three buttons onto the form to go to next \ previous page and print page. Let's use Actions collection of the ReportViewer component.

Relations between the control and Action is set via the ActionBind type methods. To bind navigation buttons to Action, it is necessary to implement ActionBind descendant.

For example, like this:

```
public class ButtonActionBind : ActionBind
{
    public ButtonActionBind(Button button)
    {
        SetComponent(button);
    }

    public Button Button
    {
        get
        {
            return this.Component as Button;
        }
    }

    protected override void Bind()
    {
        Button.Click += new EventHandler(Button_Click);
    }

    protected override void Unbind()
    {
        Button.Click -= new EventHandler(Button_Click);
    }

    public override void Update()
```



```
{
    if (Action != null)
    {
        Button b = Button;
        b.Enabled = Action.Enabled;
        b.Text = Action.Text;
        b.Visible = Action.Visible;
    }
}

private void Button_Click(object sender, EventArgs e)
{
    if (Action != null)
        Action.ExecuteAction();
}
}
```

### Bind buttons to Actions in ReportViewer:

```
PerpetuumSoft.Reporting.Windows.Forms.Action action =
reportViewer1.Actions["PrevPage"];
ButtonActionBind bind = new ButtonActionBind(prevPageButton);
action.Bind(bind);
action = reportViewer1.Actions["NextPage"];
bind = new ButtonActionBind(nextPageButton);
action.Bind(bind);
action = reportViewer1.Actions["Print"];
bind = new ButtonActionBind(printButton);
action.Bind(bind);
```

So, we bound buttons to the set actions of the ReportViewer.

Source of the generated report for the ReportViewer is set in the Source property – type implementing IReportSource interface. Constructor of the form ReportViewer is located on accepts parameter of the type implementing IReportSource interface and sets it for the Source property.

This sample is located in the CustomDesignerViewer folder.

### Custom Wizard

TBD

### Custom Report Element

Sample of the custom report element is located in the CustomReportControlDemo folder.

There created a component – descendant from the Box object with added Color property setting color and Checked property. If Checked is set to true, object area is crossed with two diagonal lines. Stored attribute sets that this property is saved when the document is written. ReportBindable attribute sets that this property can contain script calculating its value.

If you want to change default values of some class properties, you need to write a corresponding code to the InitNew() method. PaintContent method draws the component. And the main method id Render. This method is called to display component in the final report. You need to create a new instance of your object in this method.

```
CheckBoxReportControl result = new CheckBoxReportControl();
```

Then set its properties by calling a method

```
PopulateProperties(result);
```

Set its location in the final report

```
RenderLocation(result);
```

Place it on the page in the final report.





```
Engine.ProductionPage.Controls.Add(result);
```

To register the component and add it to the report template, add the following code to the Load event handler:

```
PerpetuumSoft.Reporting.DOM.ReportControl.RegisterControlType(typeof(CheckBoxReportControl));  
CheckBoxReportControl customControl = new CheckBoxReportControl();  
customControl.Location = new Point(300, 300);  
customControl.Size = new Size(300, 300);  
Document template = reportSlot.LoadReport();  
template.Pages[0].Controls.Add(customControl);  
reportSlot.SaveReport(template);
```

## Custom Export Filter

TBD

## Custom Template Storage

TBD

## Live Report

SharpShooter Reports allows you to redefine standard mechanism of handling hyperlink clicks and any other objects in the report. Sample demonstrating this feature is located in the LiveReportExample folder. Report template contains TextBox object and two Shape objects. TextBox.Hyperlink property is set to "New report". To intercept hyperlink click, HyperlinkClick event of the ReportManager component is used. Let's show new report when users click this link. HyperlinkClick event handler is shown below:

```
private void reportManager1_HyperlinkClick(object sender,  
PerpetuumSoft.Reporting.Components.HyperlinkEventArgs e)  
{  
    if (e.Hyperlink == "New report")  
    {  
        e.Handled = true;  
        try  
        {  
            reportSlot2.RenderDocument();  
            using (PerpetuumSoft.Reporting.View.PreviewForm previewForm = new  
PerpetuumSoft.Reporting.View.PreviewForm(reportSlot2))  
            {  
                previewForm.WindowState = FormWindowState.Maximized;  
                previewForm.ShowDialog(this);  
            }  
        }  
        Catch (Exception exception)  
        {  
            MessageBox.Show(exception.Message, "Report Sharp-Shooter",  
MessageBoxButtons.OK, MessageBoxIcon.Error);  
        }  
    }  
}
```

HyperlinkEventArgs class has two important properties: Hyperlink – hyperlink string and Handled – getting value that the event is handled.

When you click Shape objects, their names are shown. Use ViewObjectClick event of the ReportViewer component. This event handler contains the following code:

```
private void reportManager1_ViewObjectClick(object Sender,  
PerpetuumSoft.Reporting.View.ReportViewEventArgs e)  
{  
    if (e.Control is PerpetuumSoft.Reporting.DOM.Shape)  
    {
```



```
    MessageBox.Show("You click on shape: " + (e.Control as  
PerpetuumSoft.Reporting.DOM.Shape).Name);  
    e.Handled = true;  
}  
}
```

ReportViewEventArgs class contains two properties: Control of the PerpetuumSoft.Reporting.DOM.ReportControl type – clicked object and Handled of the bool type; this property gets if the event was handled.

## Managing Report Generation

SharpShooter Reports allows developers to change standard mechanism of report generation, if necessary. ManualBuildScript code for the corresponding Page object is required. Sample implementing manual report generation is located in the ManualBuildExample folder. This sample contains two reports: the first one contains multiplication table, the second one – catalog tree SharpShooter Reports is installed in.

Let's start from the simple sample: multiplication table. Report template contains Detail with three TextBoxes. Value property of these objects is set to a, b, c correspondingly. a, b, c are variables of the int type declared in the Document.CommonScript property. ManualBuildScript of the page contains the following code:

```
for (a = 1; a < 10; a++)  
    for (b = 1; b < 10; b++)  
    {  
        c = a*b;  
        detail1.Render();  
    }
```

So, in cycles, we set a and b variables to various pairs from 1 to 10, calculate value of c, and detail1.Render() makes Detail show in the report. We set new values for the a, b, c variables that will be displayed in the TextBox objects contained in this section.

Now let's examine catalog tree sample. Template contains 1 detail1 band with textBox1 object inside. This textbox is used to display current catalog or file. Document.CommonScript property contains the following code:

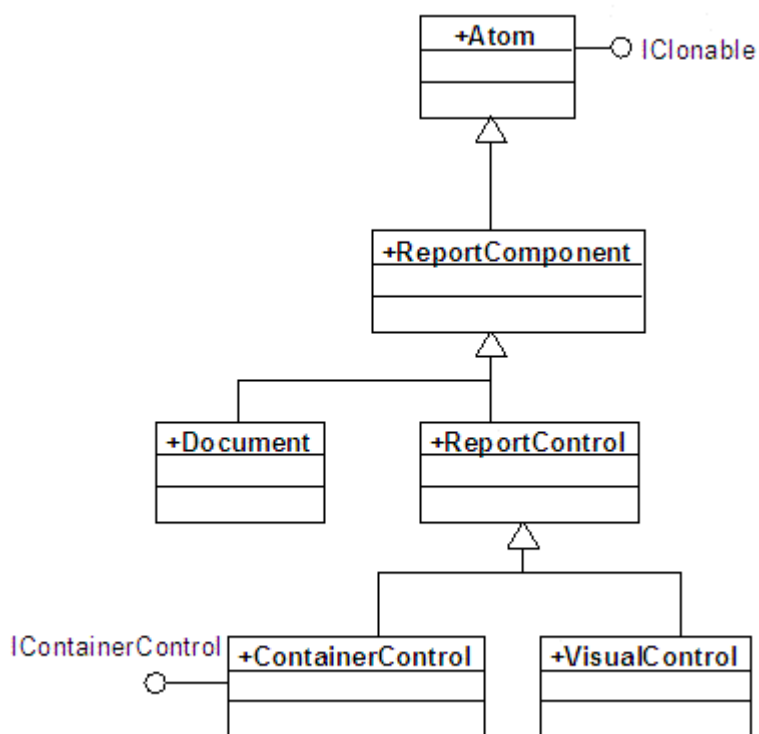
```
private string fileName;  
private float x = 0;  
private void walkTree(DirectoryInfo dirInfo, int c)  
{  
    x = (float)c * 0.5f;  
    fileName = "[" + dirInfo.Name + "];"  
    detail1.Render();  
    DirectoryInfo[] di = dirInfo.GetDirectories();  
    foreach(DirectoryInfo d in di)walkTree(d,c+1);  
    FileInfo[] fi = dirInfo.GetFiles();  
    x = (float)(c+1) * 0.5f;  
    foreach(FileInfo f in fi)  
    {  
        fileName = f.Name;  
        detail1.Render();  
    }  
}  
private void buildTree()  
{  
    string path =  
(string)Microsoft.Win32.Registry.LocalMachine.OpenSubKey("SOFTWARE\\PerpetuumSoft\\  
\\SharpShooter Reports").GetValue("Reports");  
    path = path.Substring(0,path.LastIndexOf("Examples\\Reports"));  
    DirectoryInfo dirInfo = new DirectoryInfo(path);  
    walkTree(dirInfo,0);  
}
```



Script declares two variables: fileName and x. fileName is set for the Value property of the textBox1 object, and x is used to calculate new value of the Location and Size properties of the textBox1 object. buildTree() method calculates catalog with SharpShooter Reports installed and walkTree() method is called to make recursive bypass of the catalog tree and display names of catalogs and files. In order to display catalog tree when generating report you need to add call of the buildTree() method to the Page.ManualBuildScript property.

## Appendix 1 Document Object Model

Let's consider the classes' hierarchy. These classes are used for final reports and during templates creation.



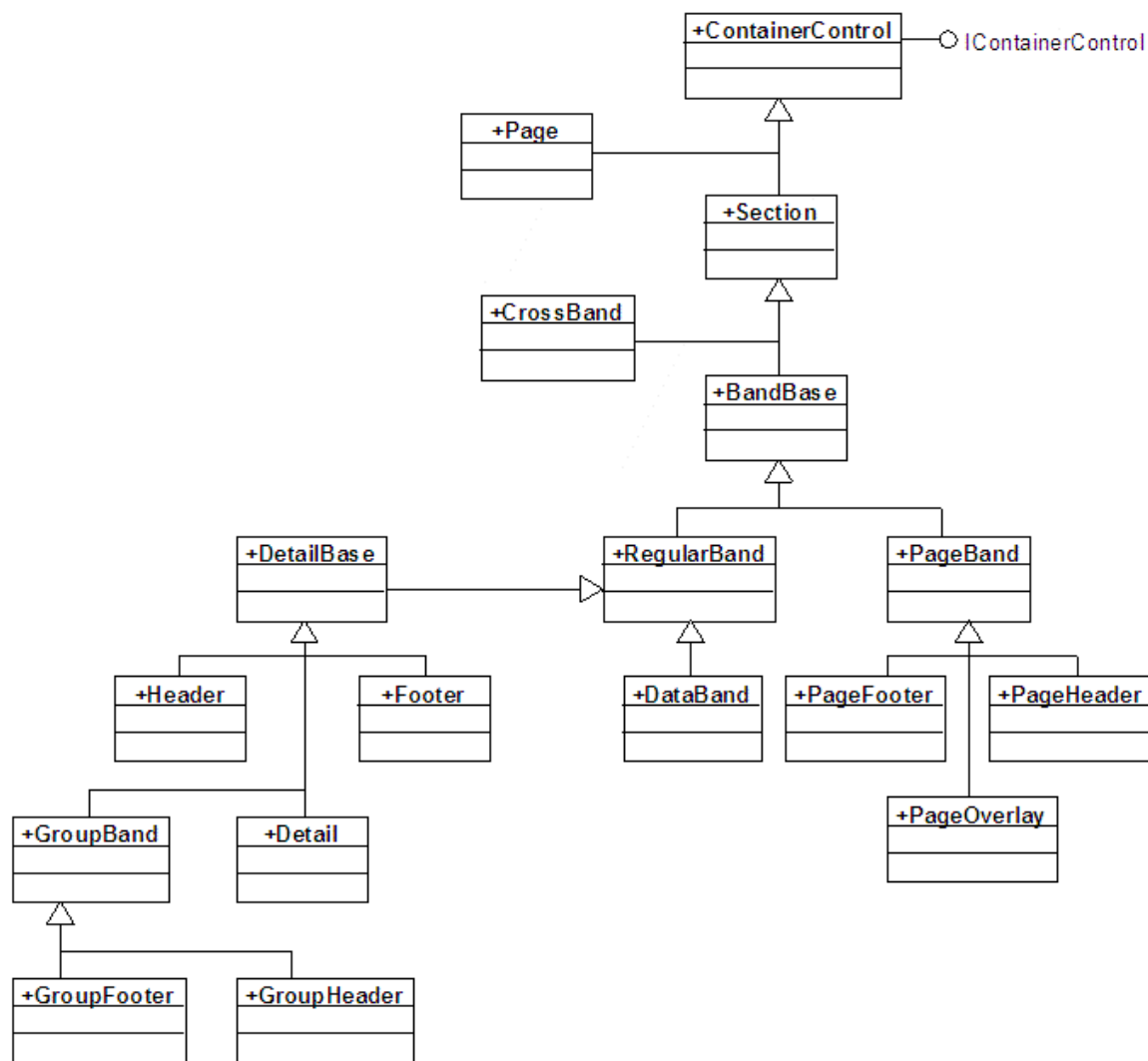
The PerpetuumSoft.Framework.Atom is a base class for all SharpShooter Reports classes. This class is a descendant from System.Object. This class realizes the System.IClonable interface.

The Atom class is a parent of ReportComponent – a base class for all SharpShooter Reports components. ReportComponent and all the classes and interfaces given below are located in the PerpetuumSoft.Reporting.DOM namespace. This class has the Name property which sets an object name.

Two classes are inherited from ReportComponent. They are Document and ReportControl. Document is used for representation of a report template and final document. It includes the Pages collection of the document pages.

ReportControl is a base class for all template elements or final document. Two classes are inherited from ReportControl. They are ContainerControl and VisualControl. ContainerControl is a base class for all non-visual report elements which, in their turn, can contain ReportControl descendants. The VisualControl class is a base class for all visual report elements.

Let's consider ContainerControl in detail. This class implements the IContainerControl interface in which the Controls property is included. This property is a collection of objects nested into this object. The IContainerControl interface includes the IsValidChild method which gets the ReportControl type parameter and sets true if the passed object can be placed into it. Below is given the hierarchy of ContainerControl classes' descendants.

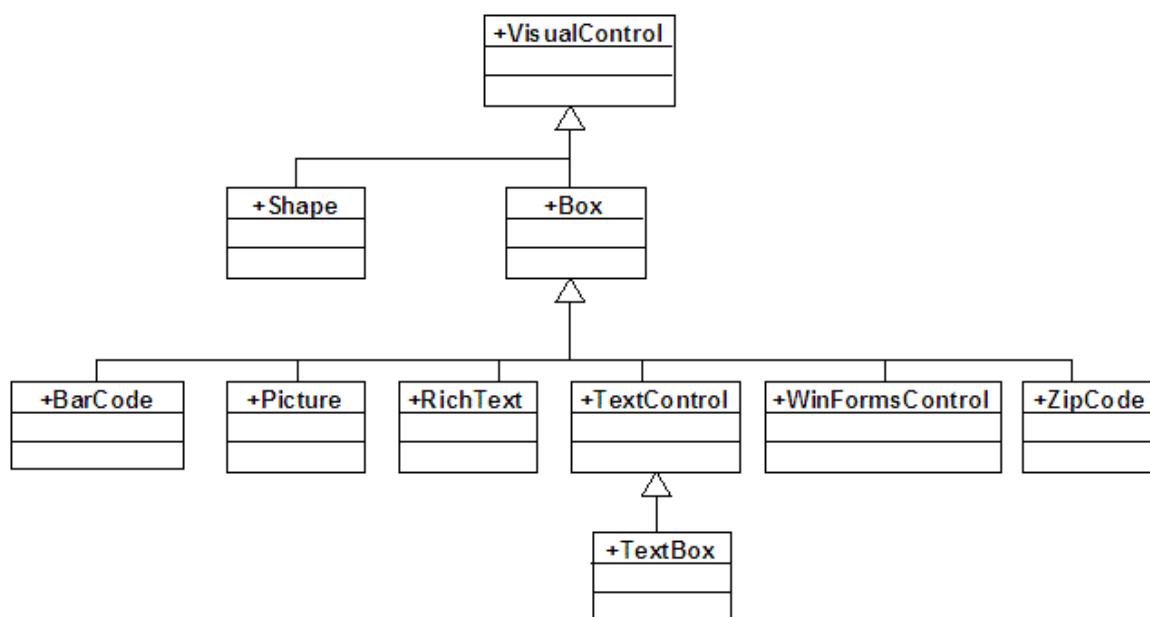


Page and Section.Page are direct descendants of ContainerControl. Page and Section.Page are pages in a template or final report.

The Section object is a basic object for all bands. CrossBand and BandBase are inherited from it. CrossBand allows the left-to-right report output. PageBand and RegularBand are descendants of BandBase. PageBand is a base class for PageHeader, PageFooter and PageOverlay sections which are output on each page. RegularBand is a base class for all other sections. It includes a collection of aggregates (the Aggregates property) and has the MasterBand property of the DataBand type. Thus, its descendants can be placed into the DataBand, including DataBand itself. This allows the creation of Master – Subreport reports with any nested levels.

DetailBase is inherited from RegularBand. This class is a base class for all sections in which visual control elements can be placed. This class has the Render method which outputs the band content into report. Header and Footer are inherited directly from DetailBase. Header and Footer are headlines for the DataBand. The Detail is inherited from DetailBase. This section is output for each record from a data source and GroupBand. GroupBandBase is a basic class for GroupHeader and GroupFooter. It has the Group property which sets an expression. If the expression is changed, the visual elements, which are included in GroupHeader and GroupFooter, will be output in the final document.

Let's consider the VisualControl class.



The Shape class is inherited from the VisualControl class. Shape allows the output of various shapes into a report. The Box class is also inherited from the VisualControl class. This class is a base class for control elements which are located in the rectangle area. It has Border, Fill and Margin properties for setting borders of an area. BarCode, Picture, RichText, WinFormsControl, ZipCode are inherited from the Box class. BarCode is a control element for bar codes output, Picture is a control element for images output, RichText is a control element for output of the text in the RTF format, WinFormsControl is a control element for Windows Forms control element output, ZipCode is a control element for zip codes output. The TextControl class is inherited from the Box class. TextControl is a base class for output of text data into a report. It has the Angle property which allows the setting of the rotation angle for text, CanGrow and CanShrink properties which allow the change of height of an object depending on the output text automatically, the Font property which sets the font, the TextAlign property which sets text alignment, the TextFill property which sets the text fill and the TextFormat property which sets the text format. TextBox is inherited from TextControl. The TextBox class has the Text value which sets a text static value; and the Value property which sets the expression for text calculation during the generation. The TextBox class has the FindText method which can be used for searching text information.



## SharpShooter Reports.Silverlight

### Getting Started

#### Introduction

The purpose of the user manual is to demonstrate main points of using SharpShooter Reports.Silverlight, provide necessary knowledge to get started with the component. You will get step by step recommendations on how to create a web application using SharpShooter Reports.Silverlight. You'll see how to create and configure a service, design a report and, at last, how to integrate a report viewer into the application pages.

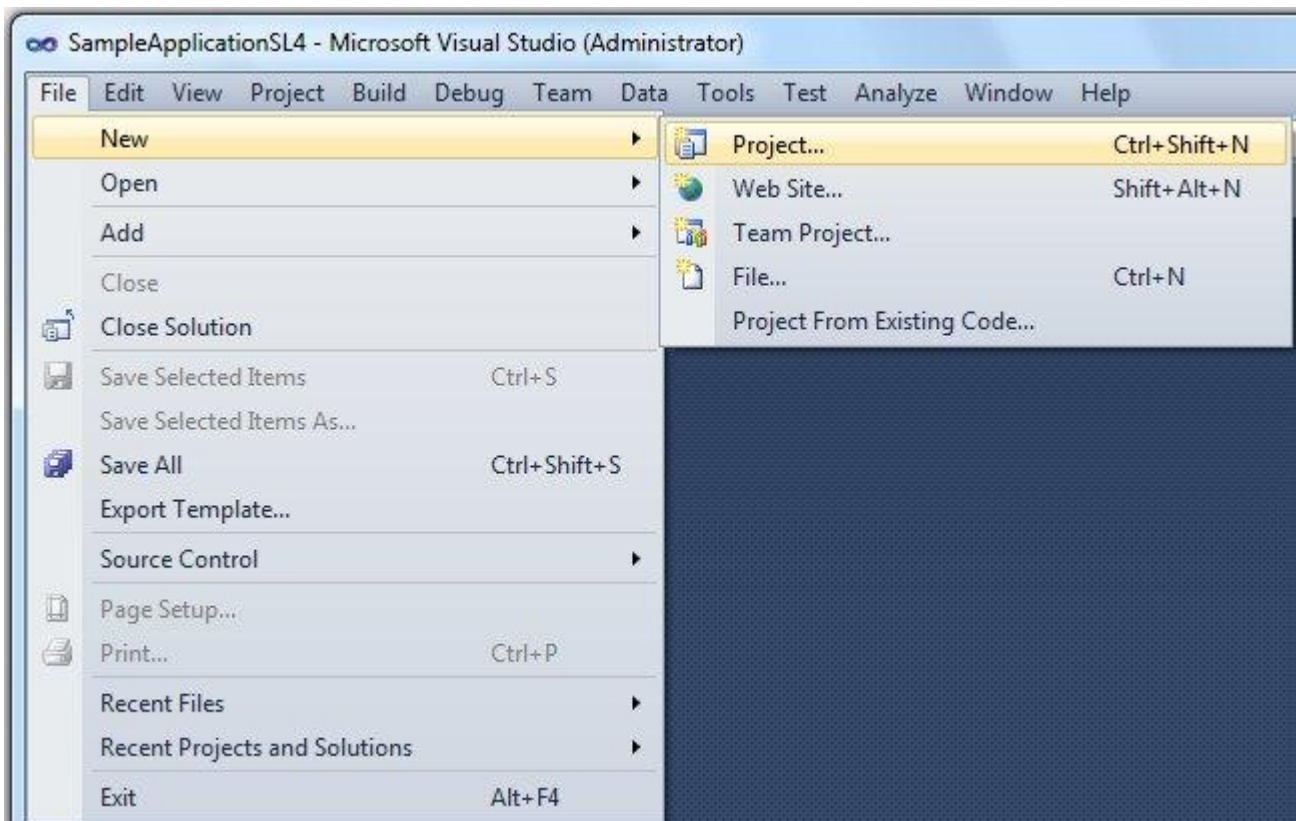
#### Product Prerequisite

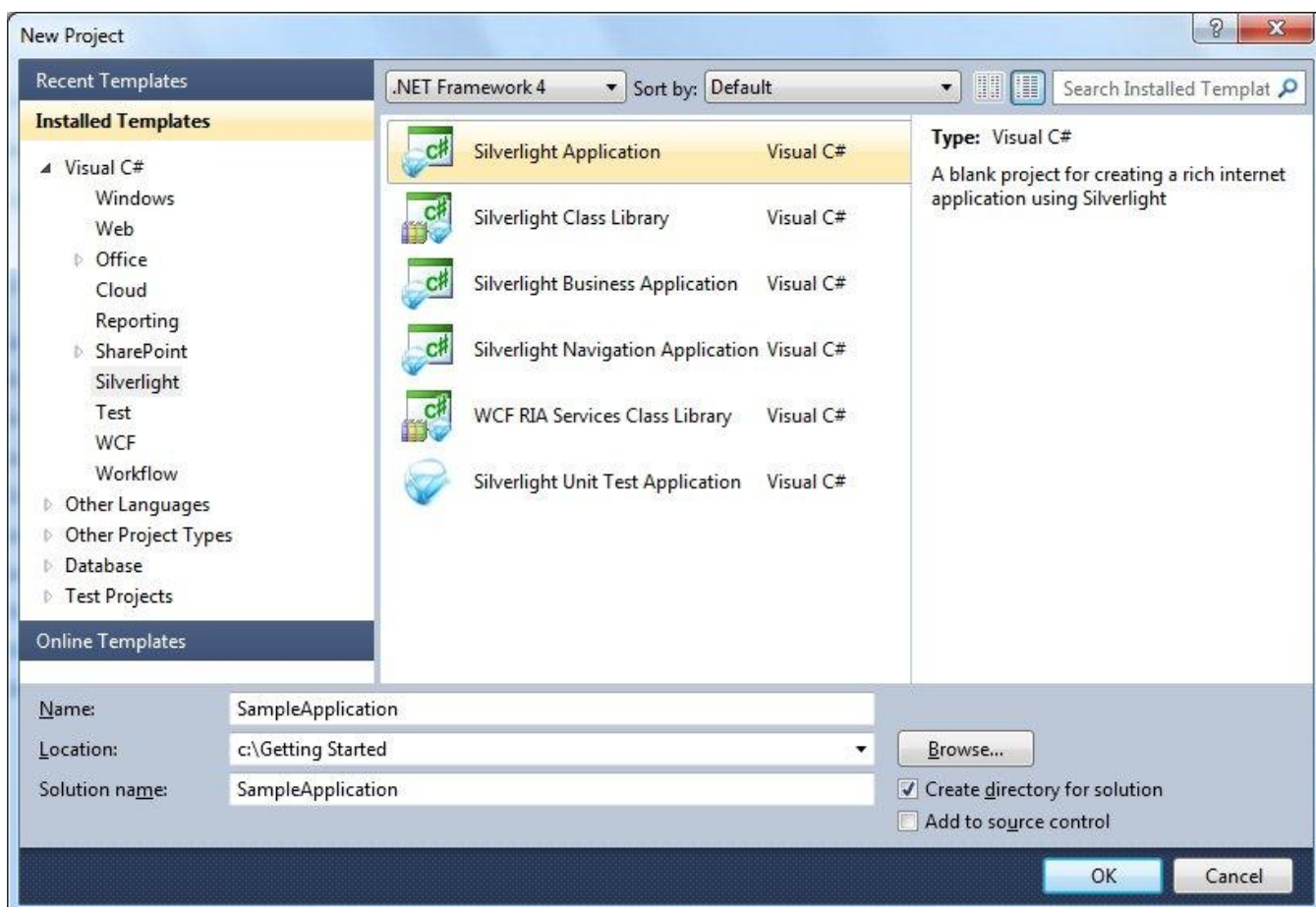
In order to get the sample working the following software should be installed on your computer:

- MS Visual Studio 2010
- .NET Framework 3.5
- ASP.NET 2.0
- Silverlight 4.0
- Microsoft Silverlight 4 Toolkit
- SharpShooter Reports.Silverlight 6.0.0.0 or higher.

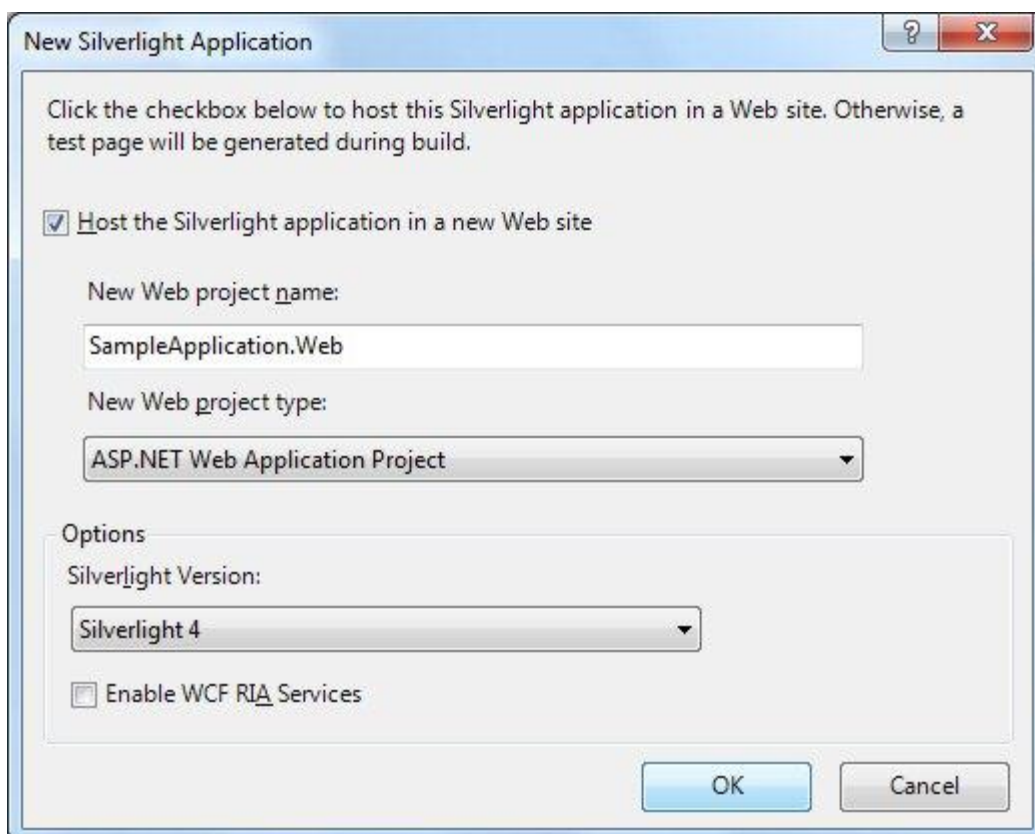
#### Creating Sample Application

**Step1.** Create a new Silverlight Application project named SampleApplication.





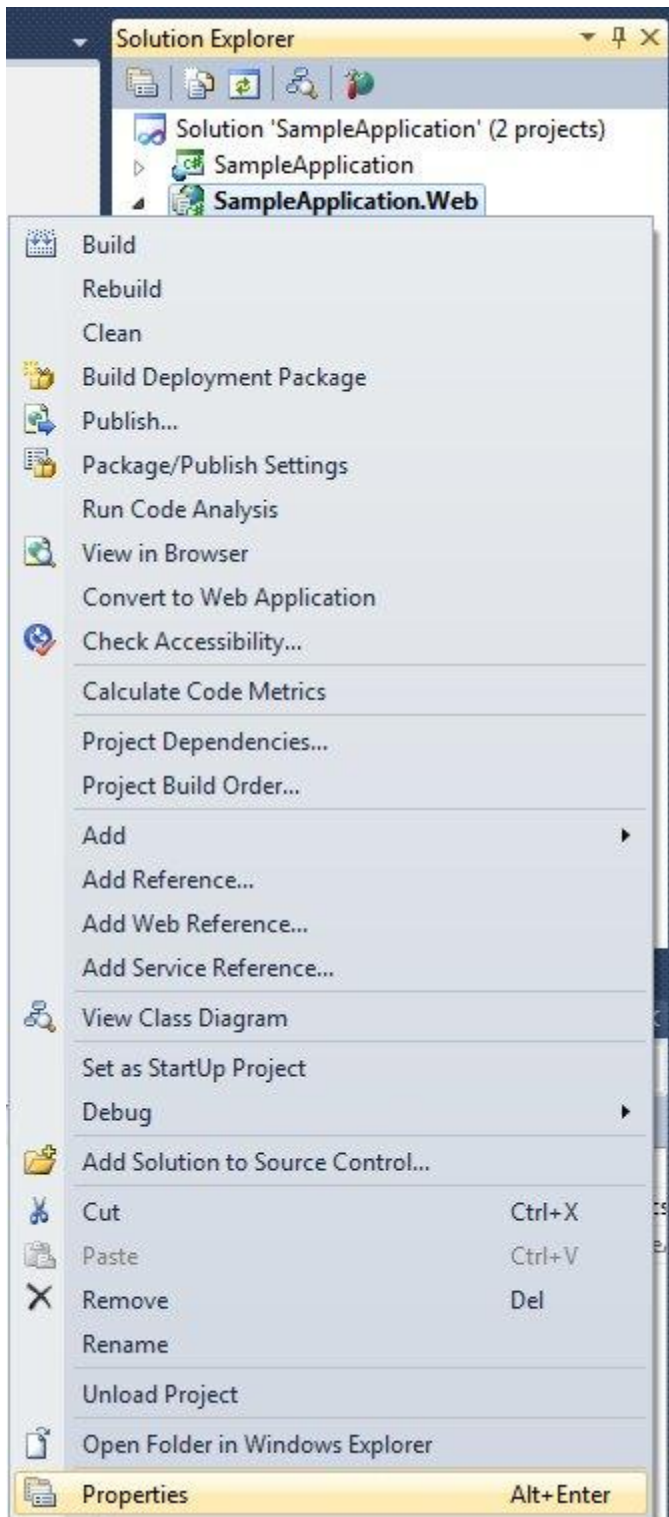
While creation of the project check option "Host the Silverlight application in a new Web site" and set name of the web-project to SampleApplication.Web



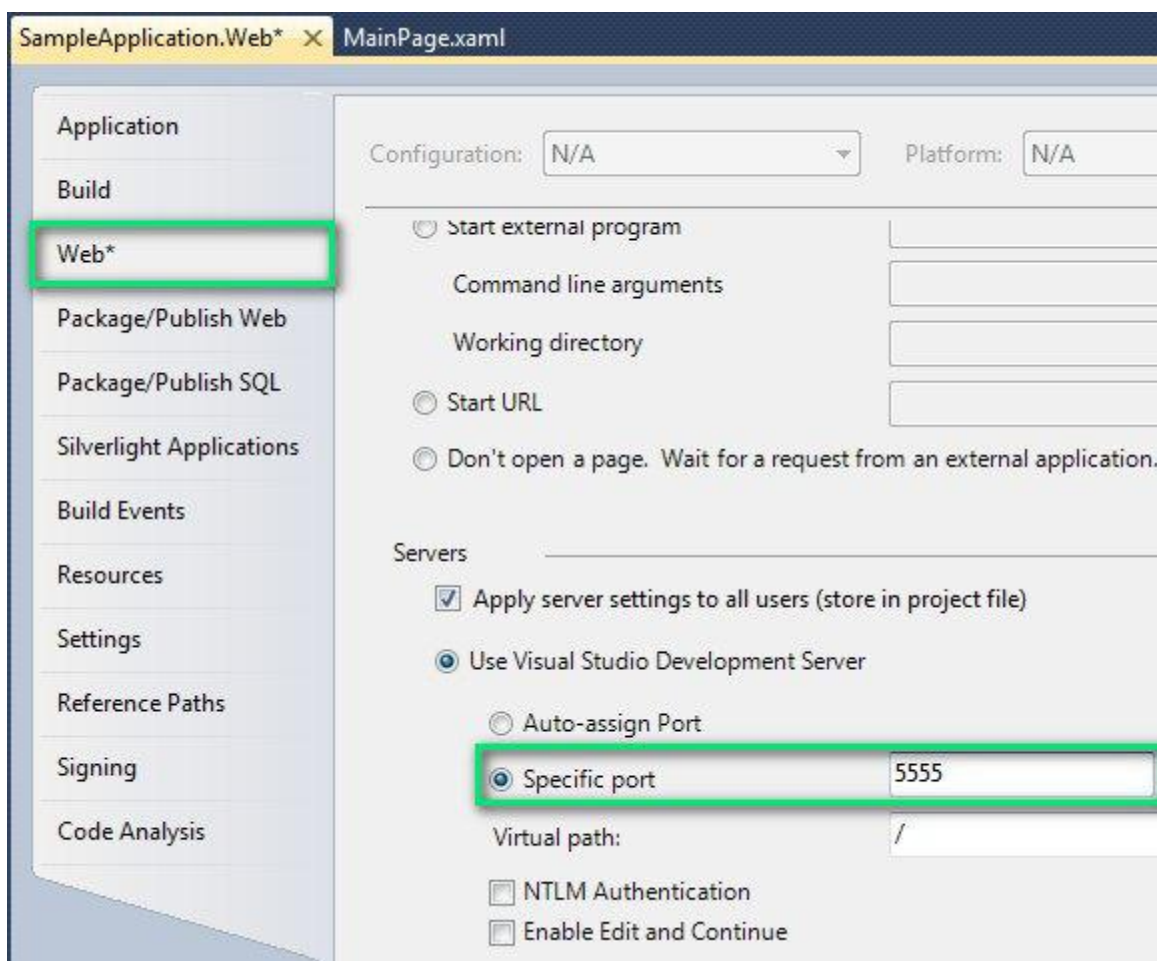


After that VS will create a new solution containing two projects.  
**Step2.** Set the web application to use static port 5555.

Open SampleApplication.Web properties

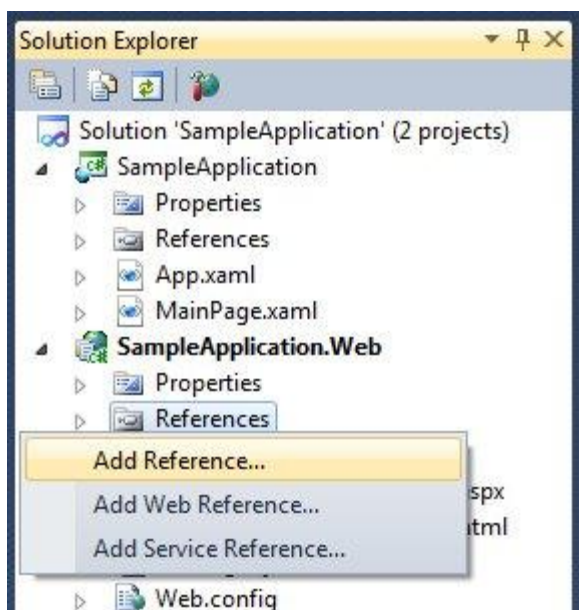


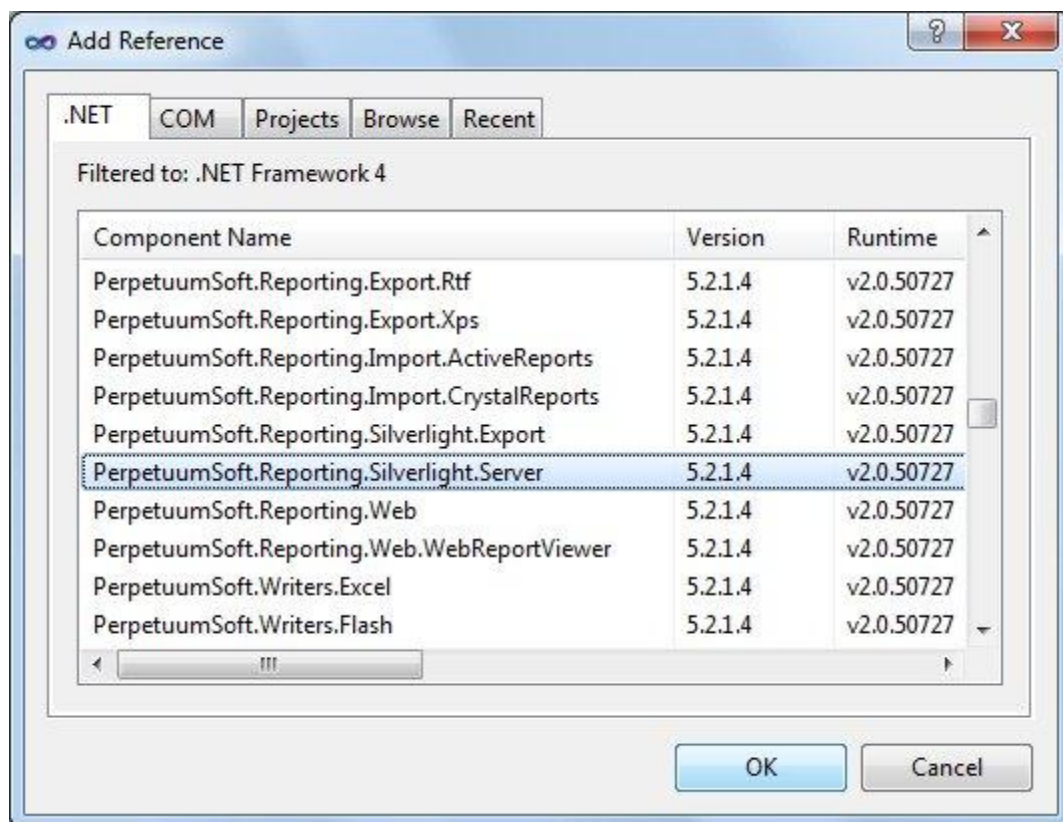
Check option Specific port and set it to 5555 in the Web tab of the SampleApplication.Web properties.



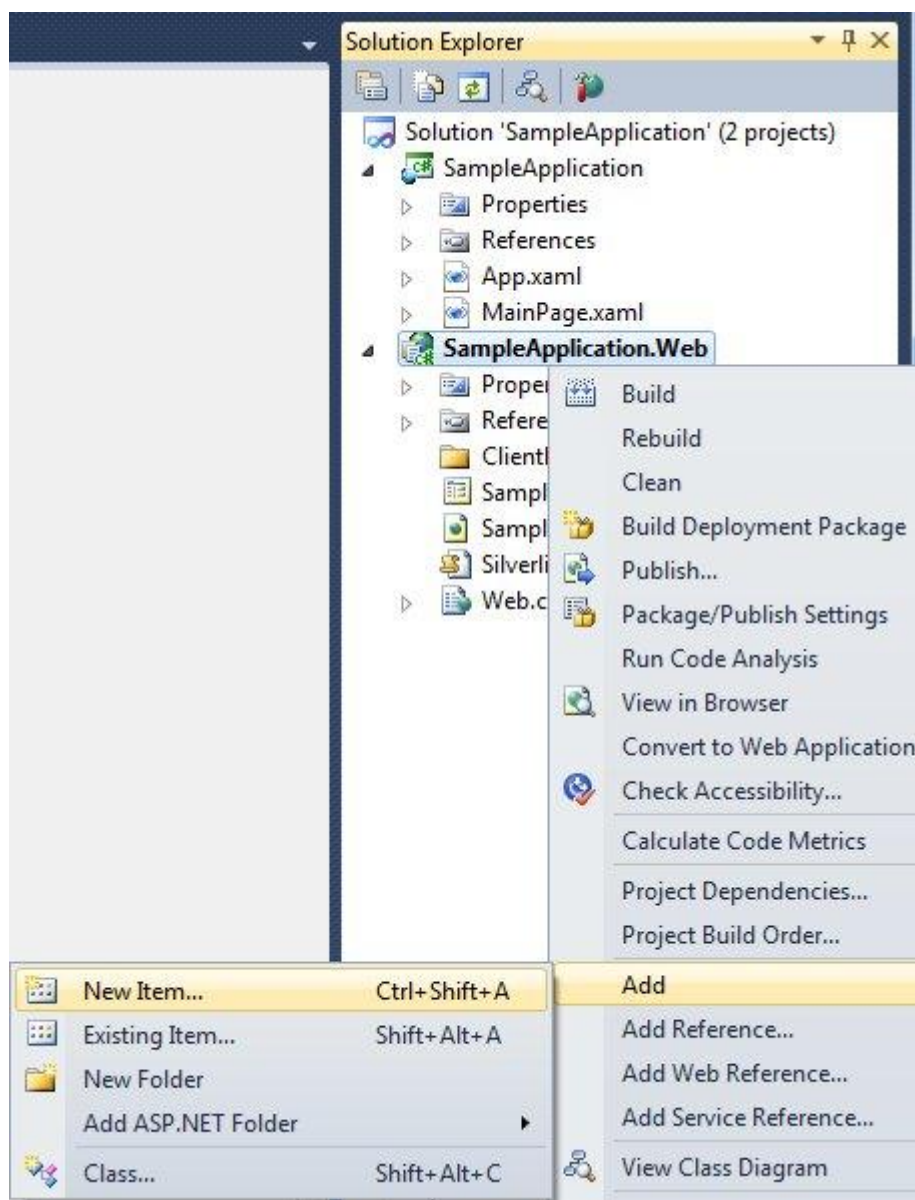
**Step3.** Adding and setting up WCF-service.

In order to make client interact with the server, we need a special service. Before you create this service you need to add reference to PerpetuumSoft.Reporting.Silverlight.Server.dll by right-clicking References node at SampleApplication.Web project and choosing Add Reference in the popup menu.

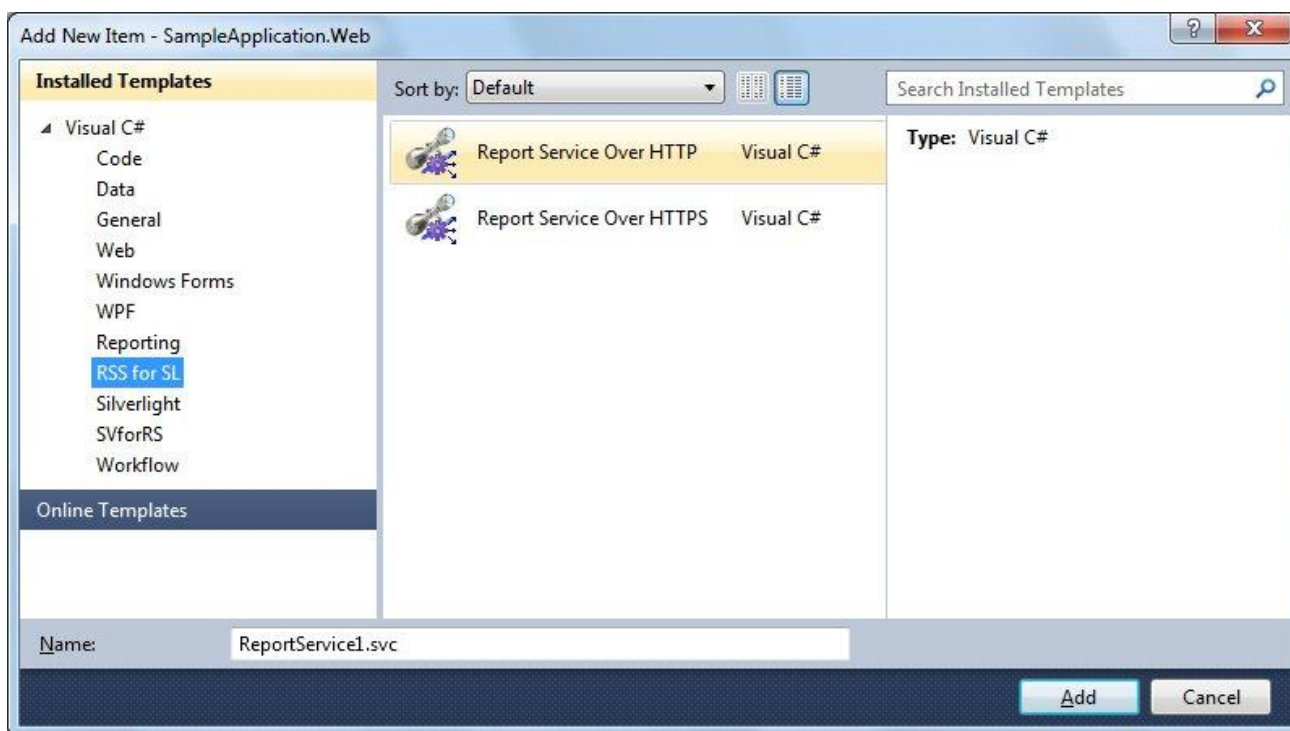




Add Report Service Over HTTP named ReportService.svc to the SampleApplication.Web project. In order to do that, right-click SampleApplication.Web and choose Add->New Item in the popup menu.



Add Report Service Over HTTP template to your project.

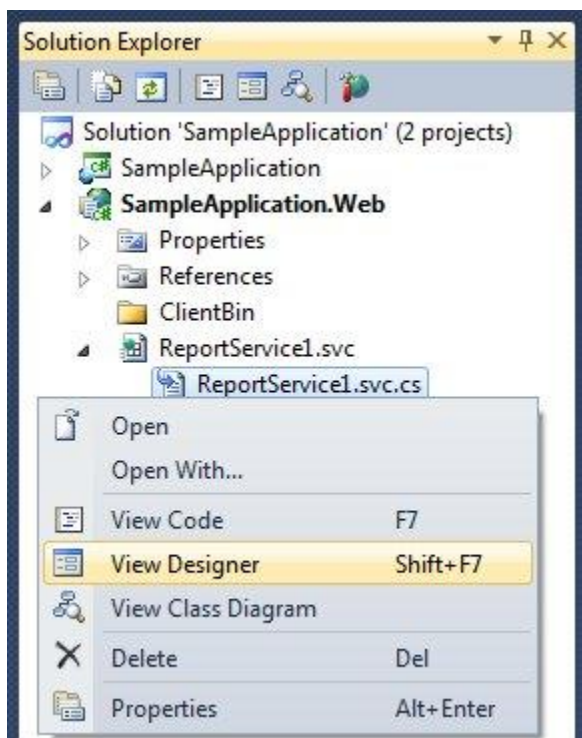


ReportService class will extend the PerpetuumSoft.Reporting.Silverlight.Server.ReportServiceBase class containing implementation of wcf-service for Silverlight ReportViewer.

**Step4.** Creating report template.

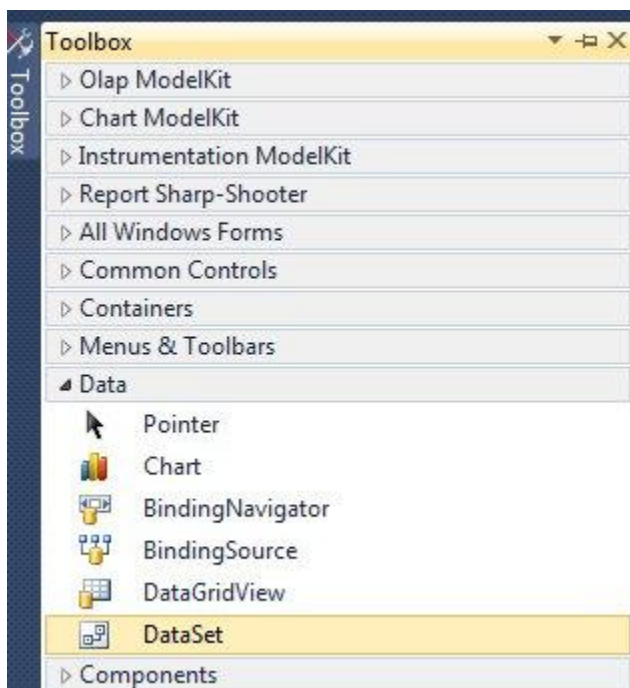
First of all we need to create data source structure. Open service in the designer.

In order to do that right-click ReportService.svc.cs and choose View Designer in the popup menu.

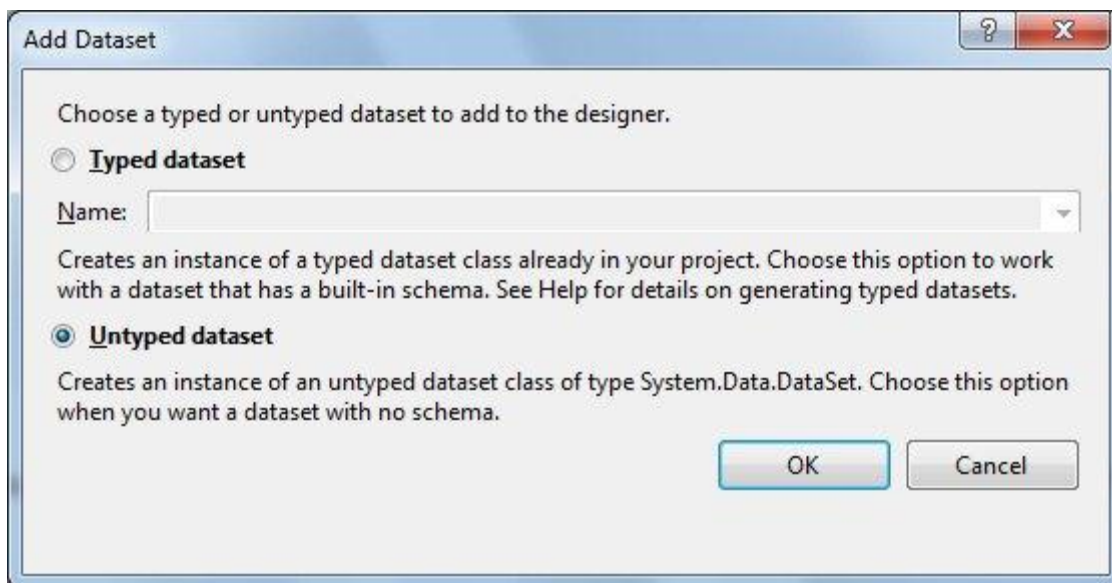


Add Dataset from the Toolbox (double click on the Dataset in Toolbox).

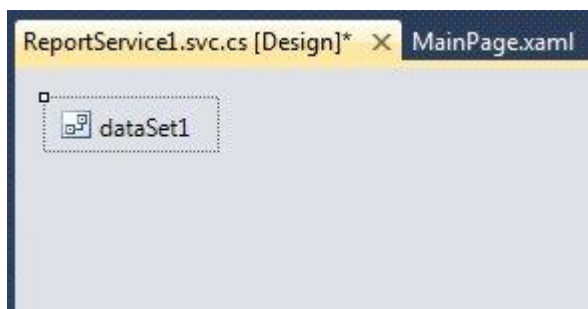




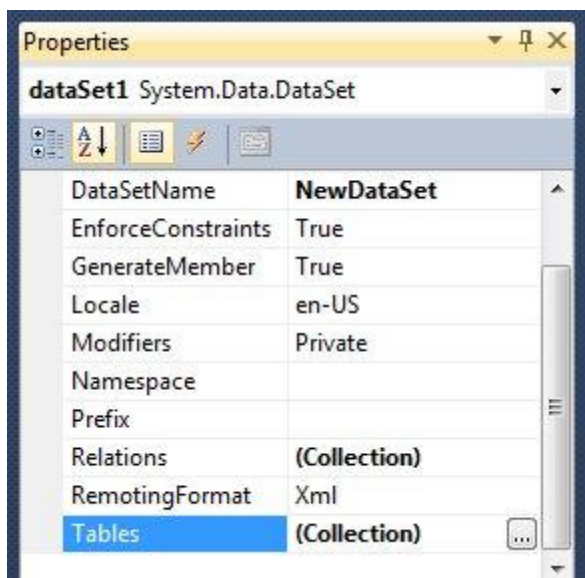
And select the Untyped dataset.




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

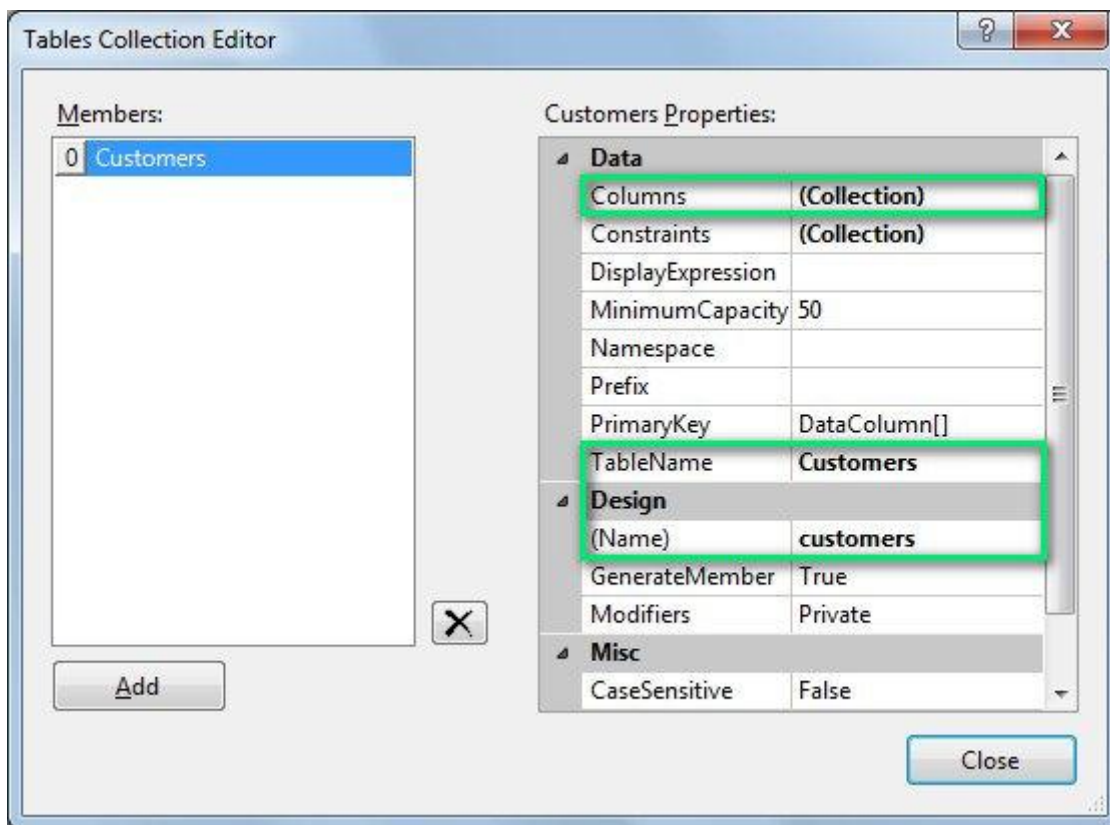


Open Tables Collection Editor (click button  in property Tables).

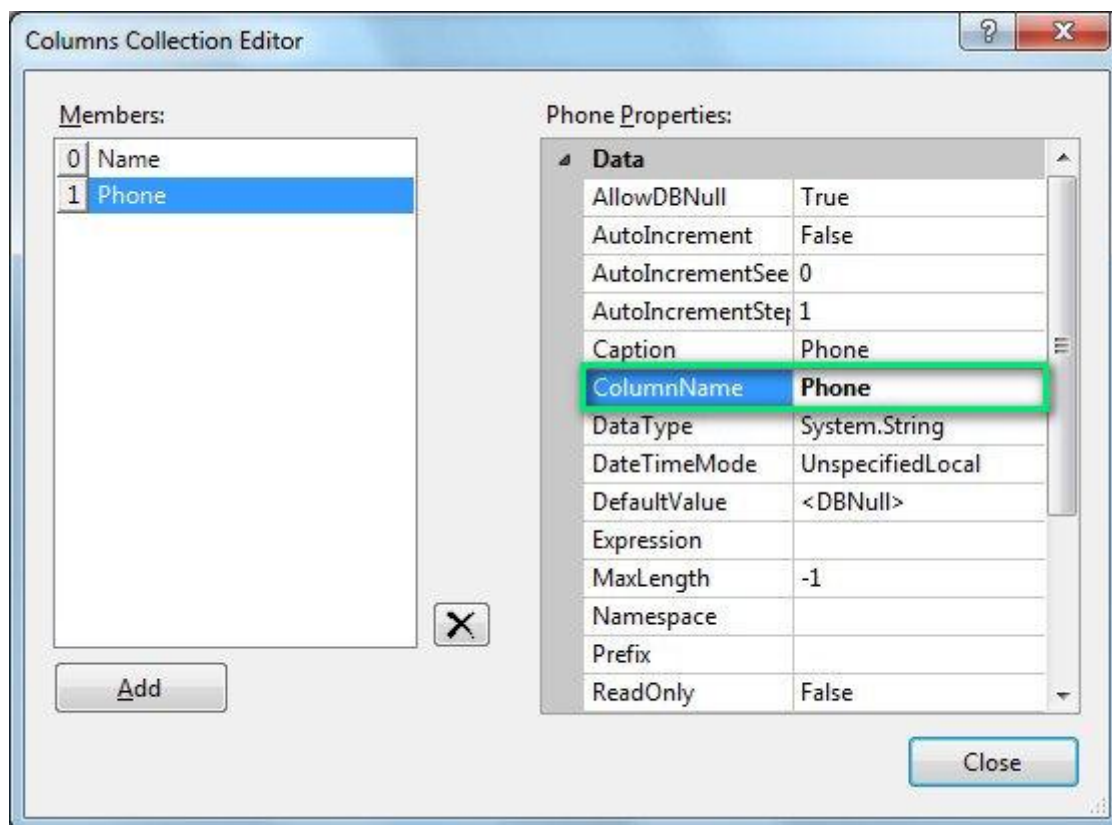


Add Customers table to the dataSet1 (click "Add" button and set TableName property value to "Customers", set Name property value to "customers").

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

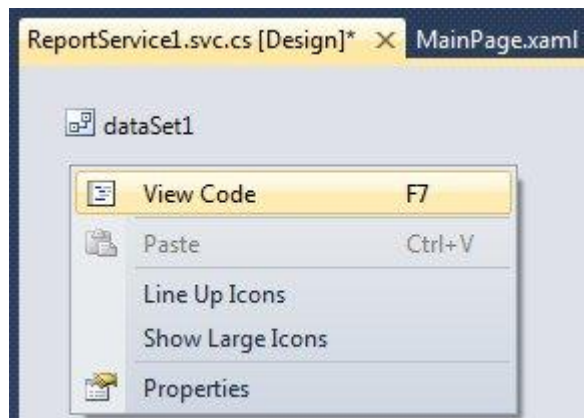


Add two columns by clicking "Add" button and setting ColumnName property value to "Name" and "Phone".



Data structure is defined, and it's necessary to fill Customers Table with the data.

To view source code right-click on designer area and click "View Code" item in contextual menu.



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

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

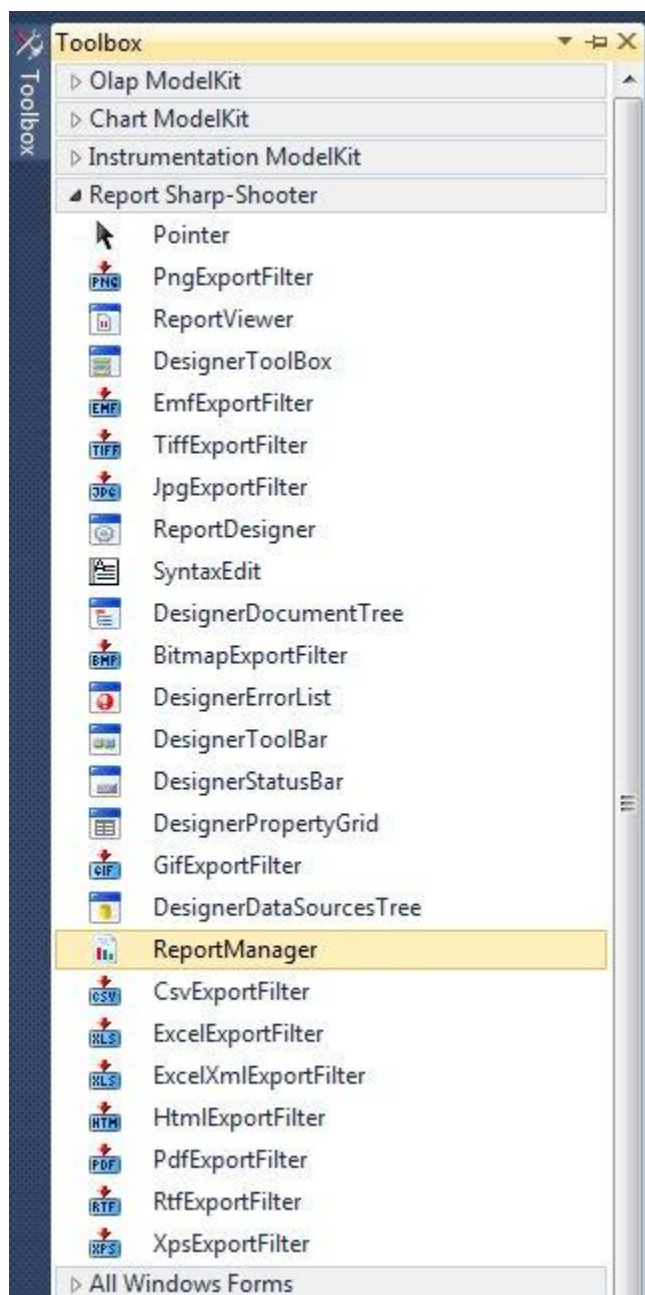




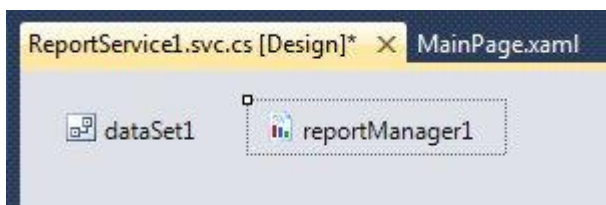
```
row = customers.NewRow();  
row["Name"] = "Brown Kelly";  
row["Phone"] = "803-438-2771";  
customers.Rows.Add(row);  
}
```

Note: Information about the queried document and its parameters is passed in the method's parameters.

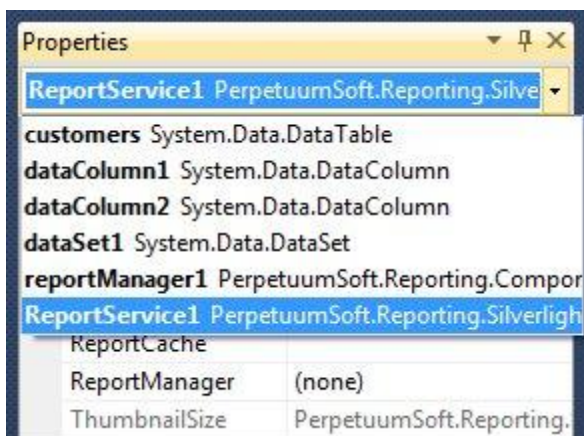
Now add the ReportManager component (double click on "ReportManager" in ToolBox); this component is responsible for report generation.



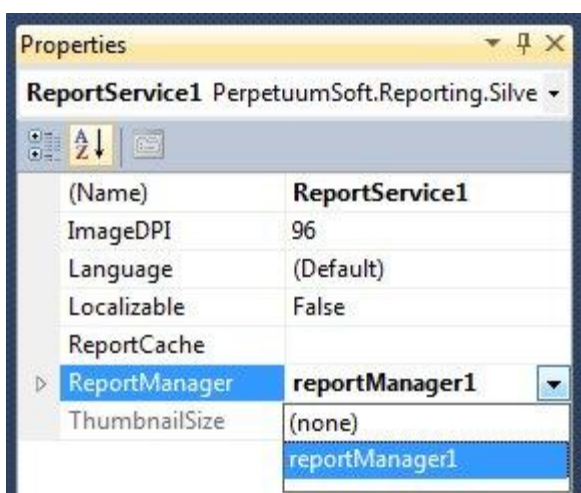
After that the ReportManager node (reportManager1) will appear in the designer.



Set the ReportManager property of the ReportService service. In order to do that, open ReportService properties in the Properties window.

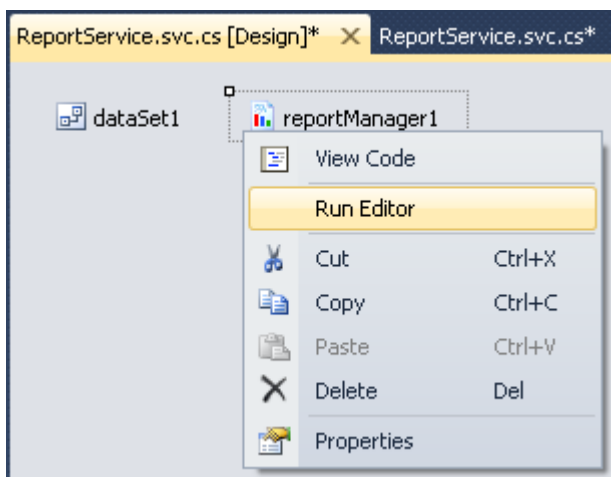


Select reportManager1 from the list:



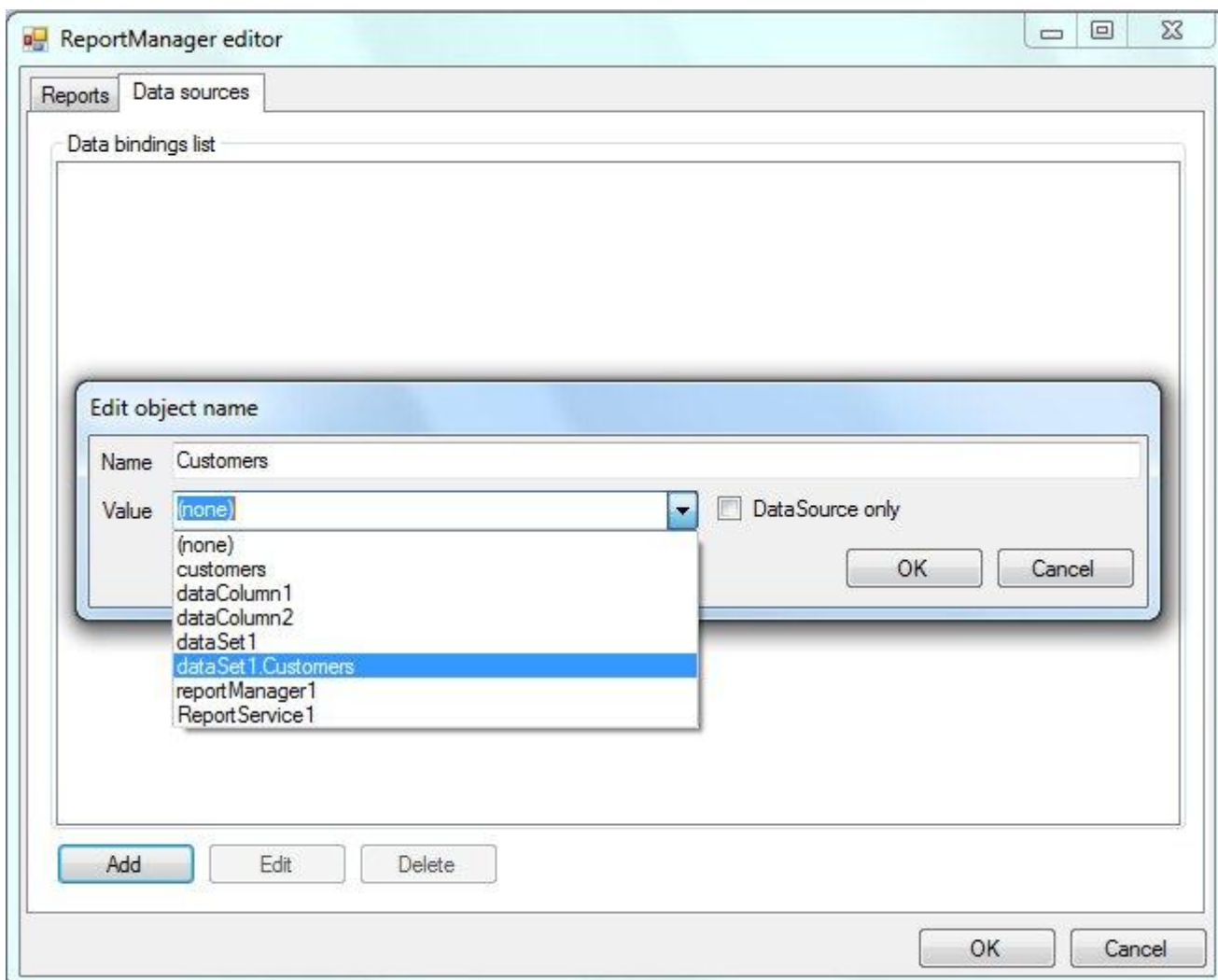
Now you should create the report.

Run Report Manager Editor by right-clicking reportManager1 and choosing Run Editor

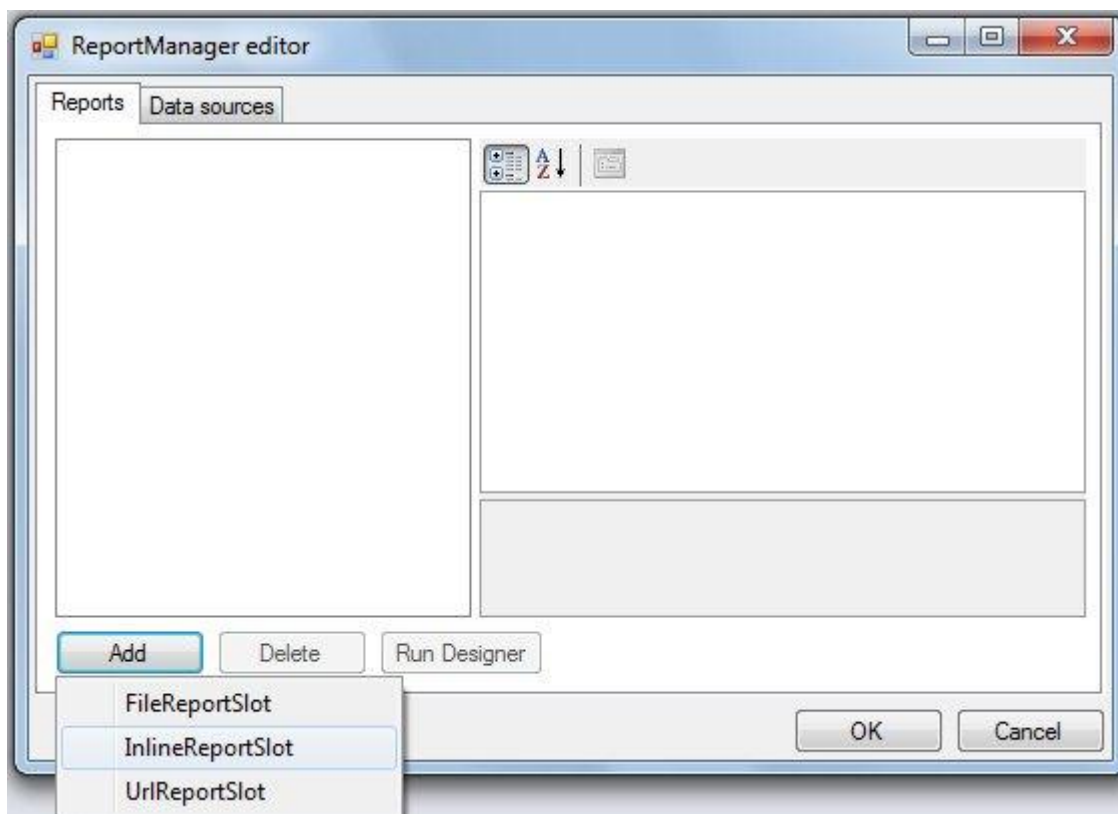


Before you start creating report template add a data source by which the report will be generated.

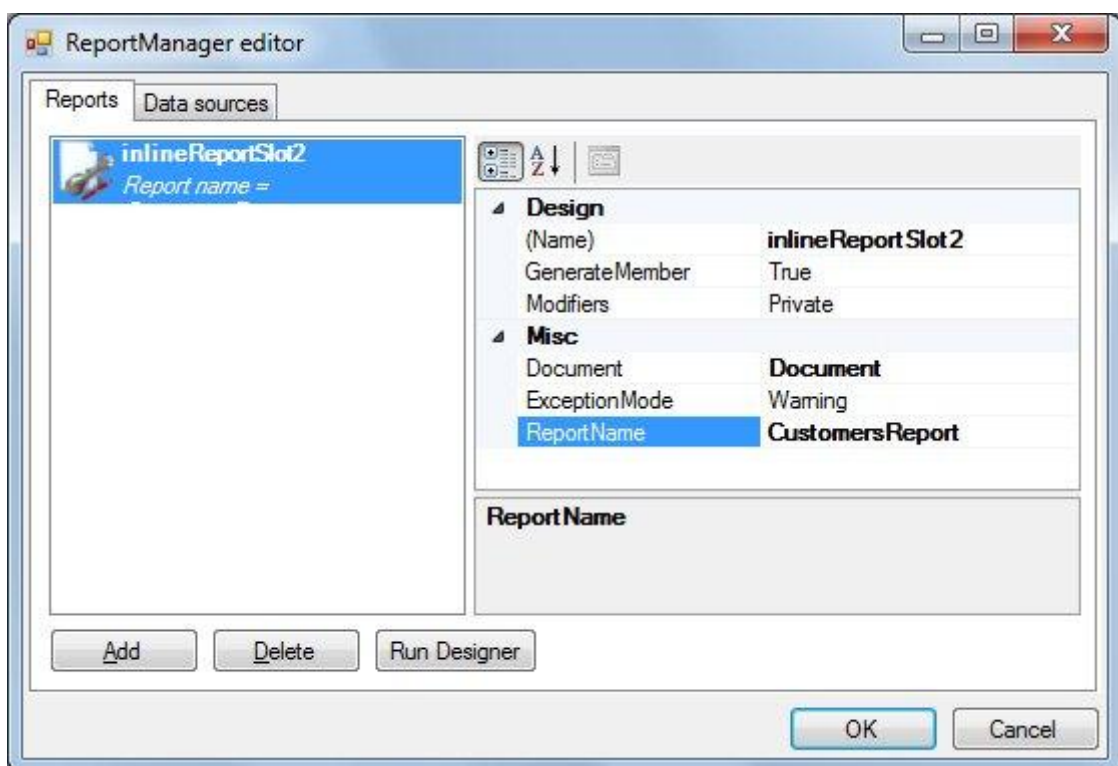
Add Customers table to the Data binding list located on the "Data sources" tab (click "Add" button, in appeared form "Edit object name" set Name value to Customers, select dataSet1.Customers in the combo box list).



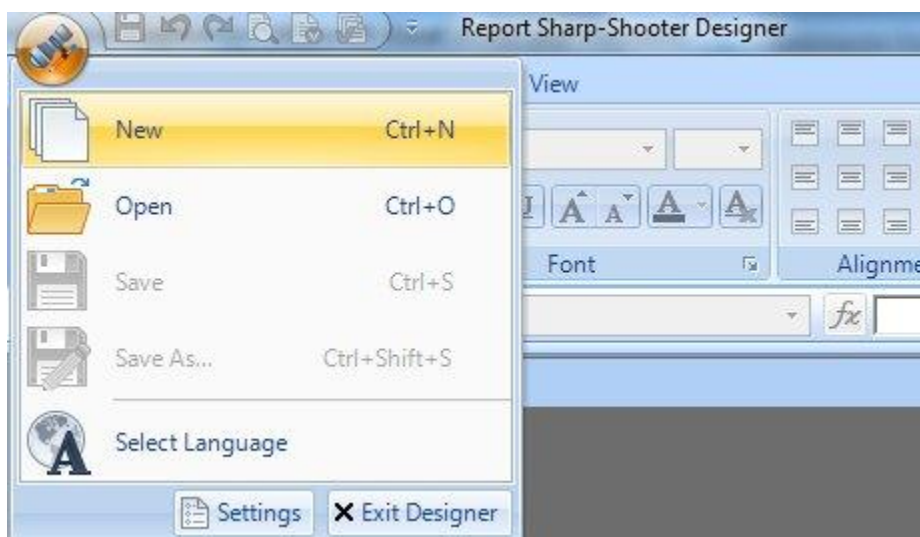
In the "Reports" tab add a new object - "InlineReportSlot" by clicking the "Add" button.



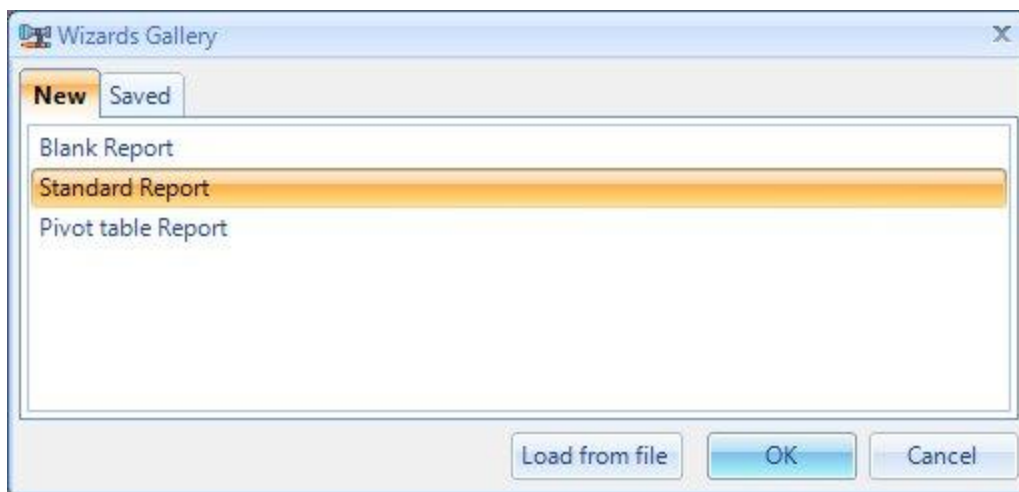
Set the ReportName property value to CustomersReport. Afterwards you will get the required document from the Report Manager exactly by that name. Then press the "Run Designer" button to launch report designer.



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

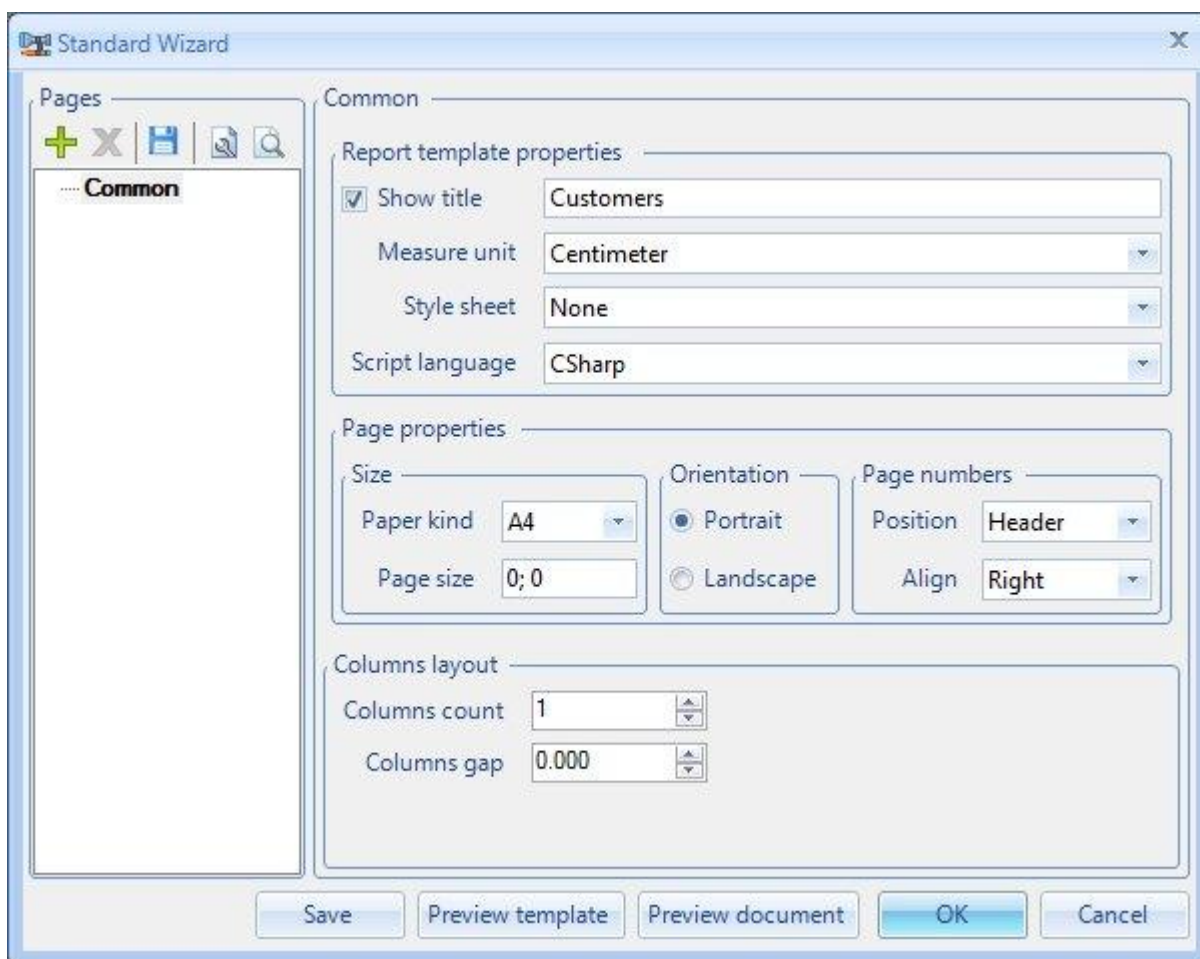


Select Standard Report in the list under the "New" tab and press OK.

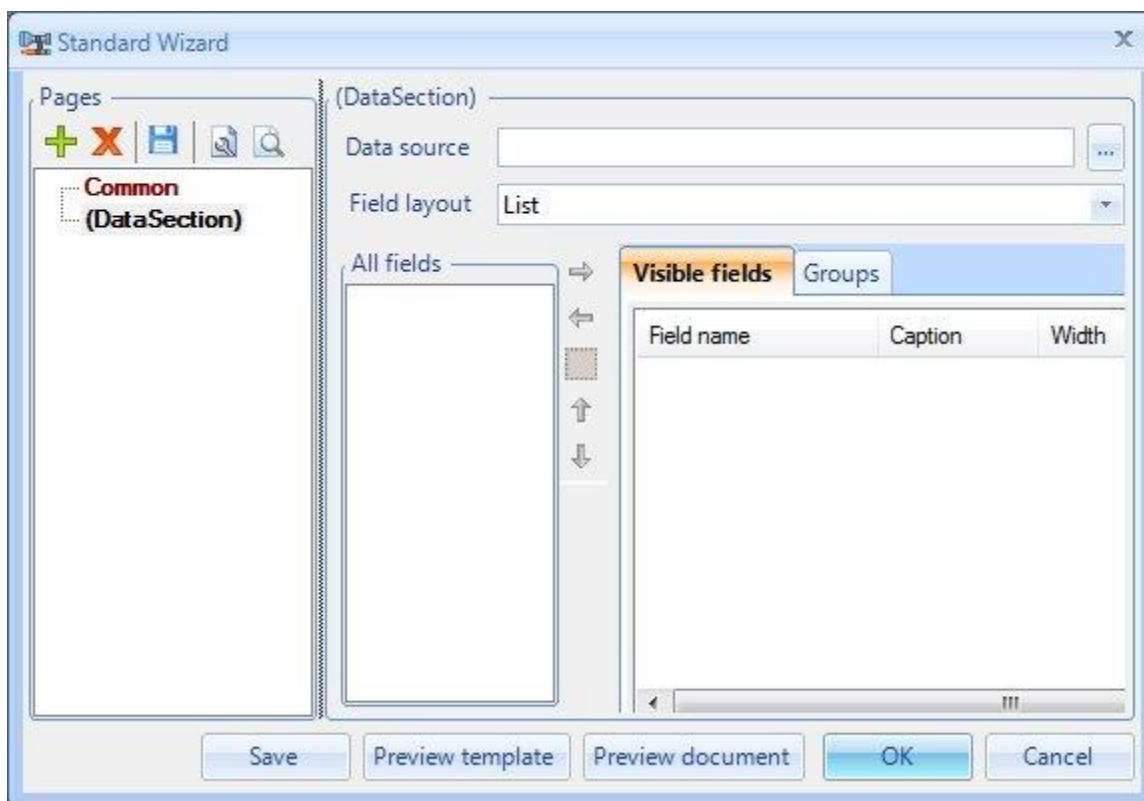


The Standard Wizard window will appear on the screen.

Set document parameters as shown in the following figure.



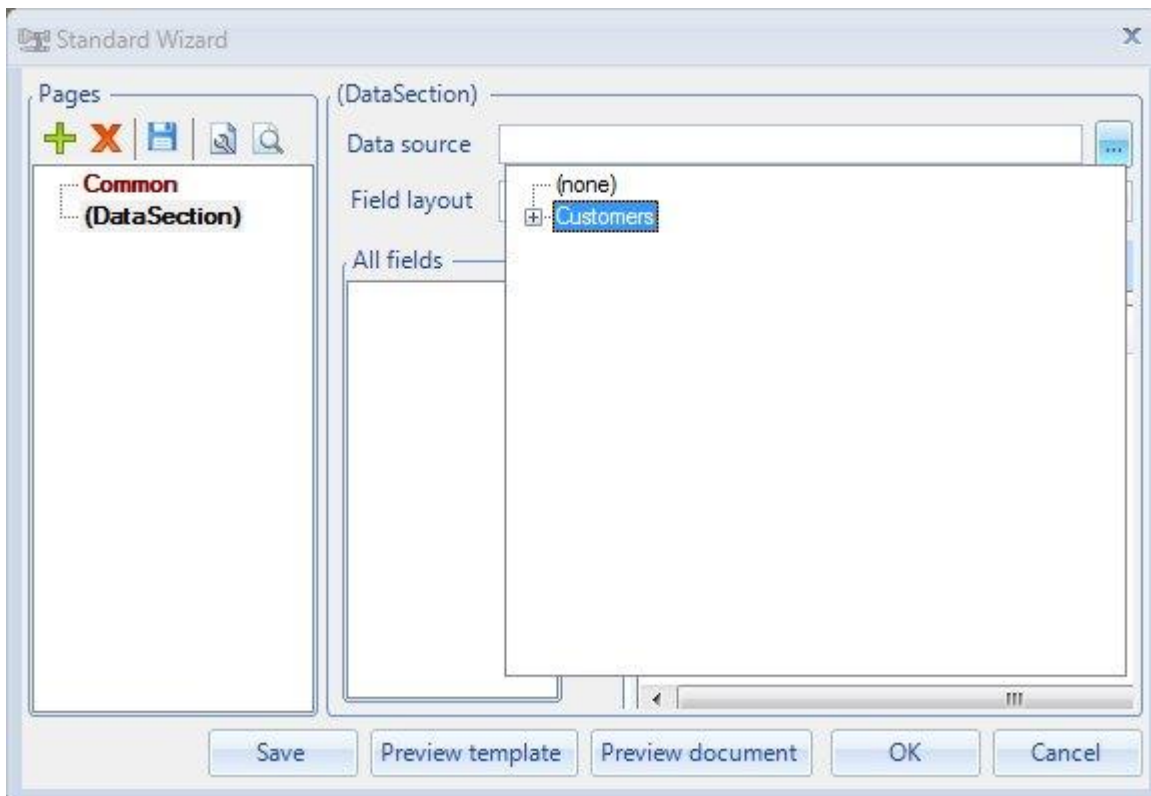
Add data source using the "Add" button (+).



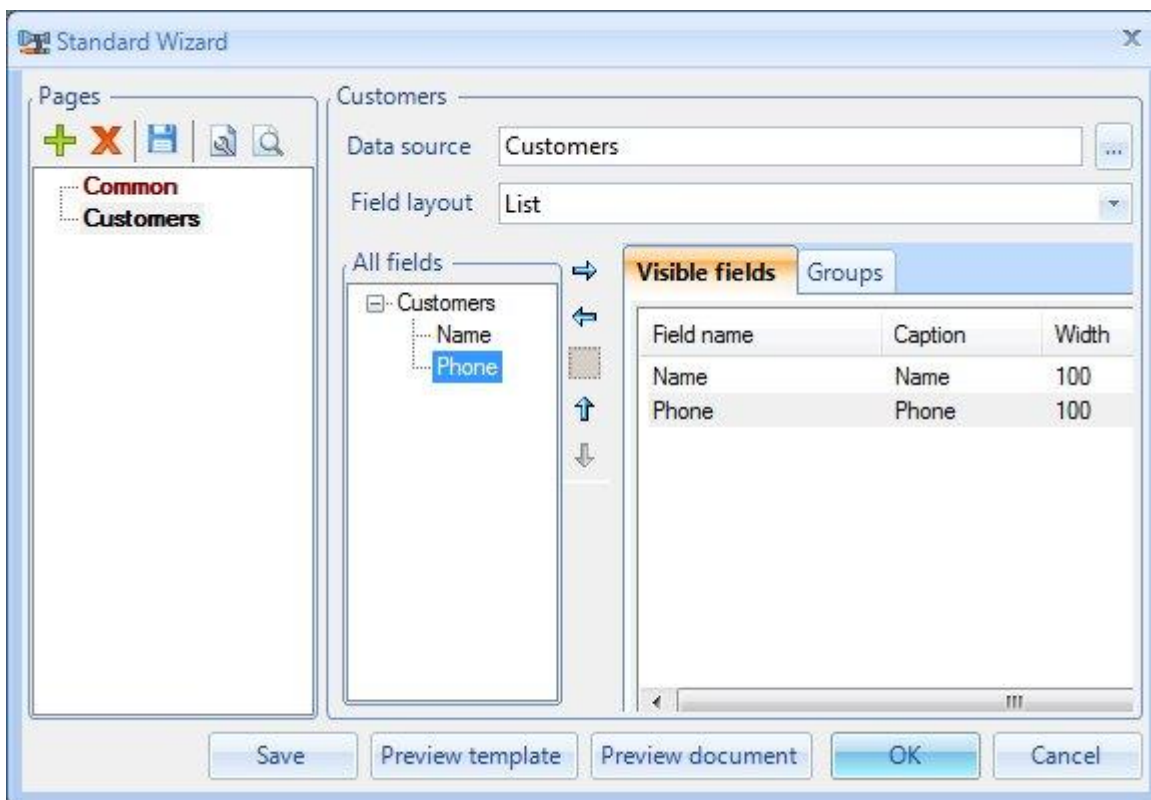




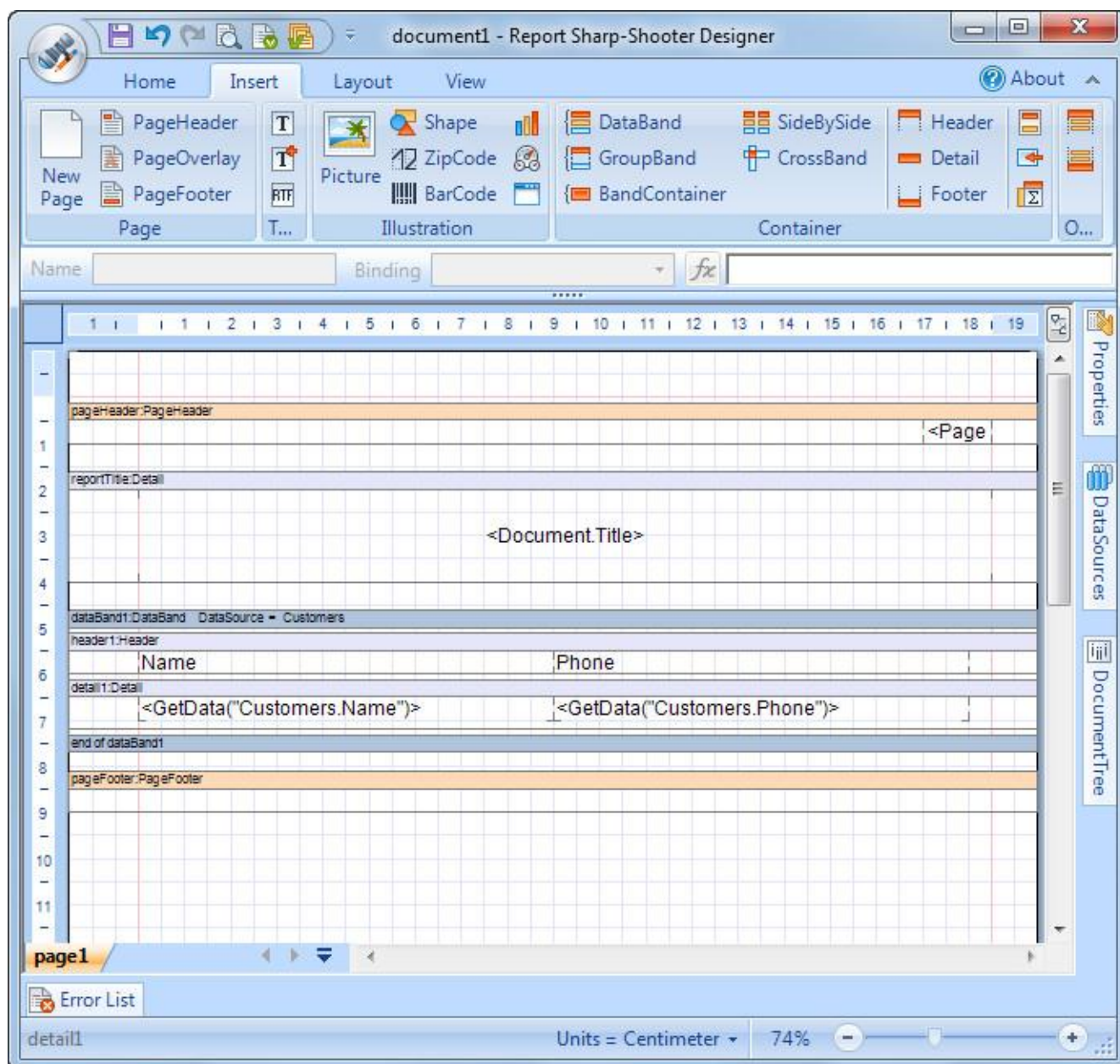
Click  button, in appeared tree view select "Customers" by double click.



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



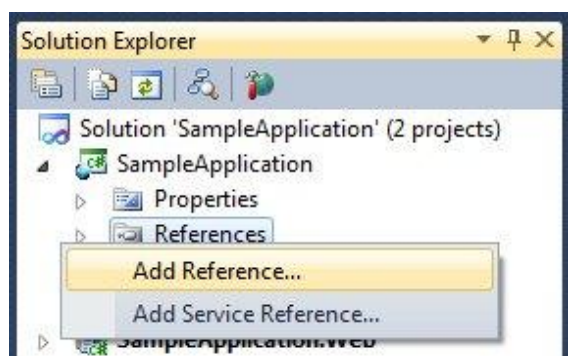
Press "OK". The template is created.



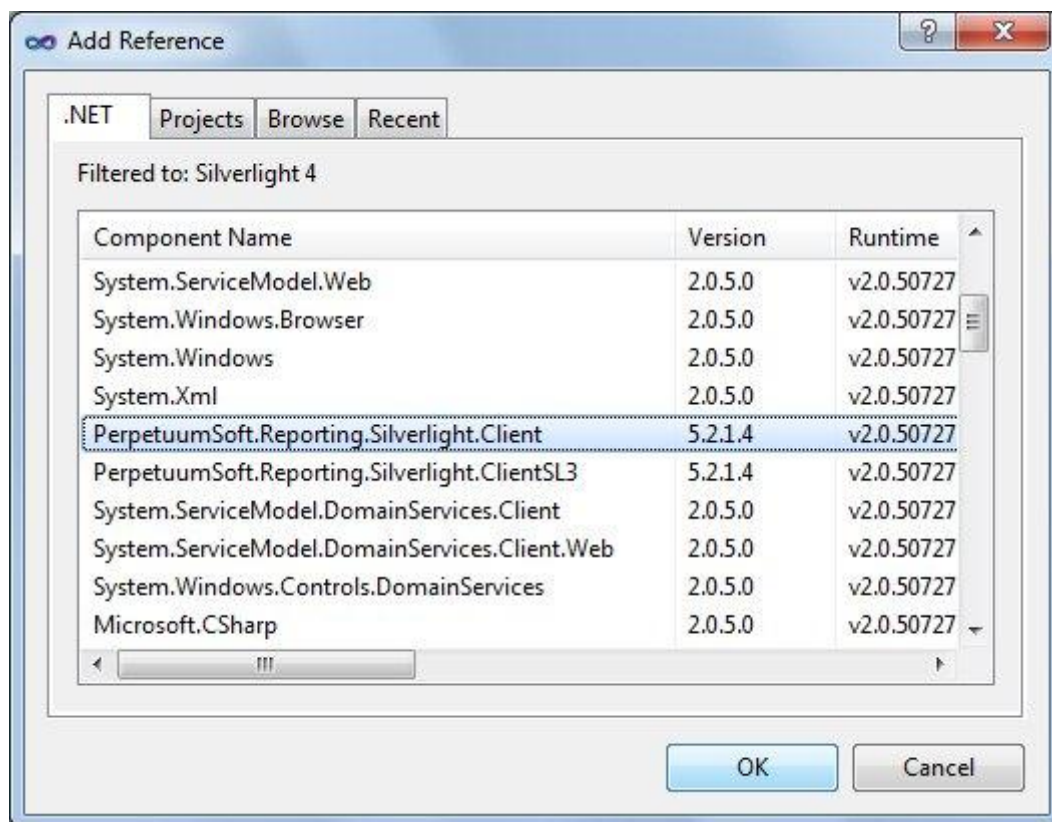
Now you should save template and close designer.

**Step5.** Adding and setting up Silverlight ReportViewer component.

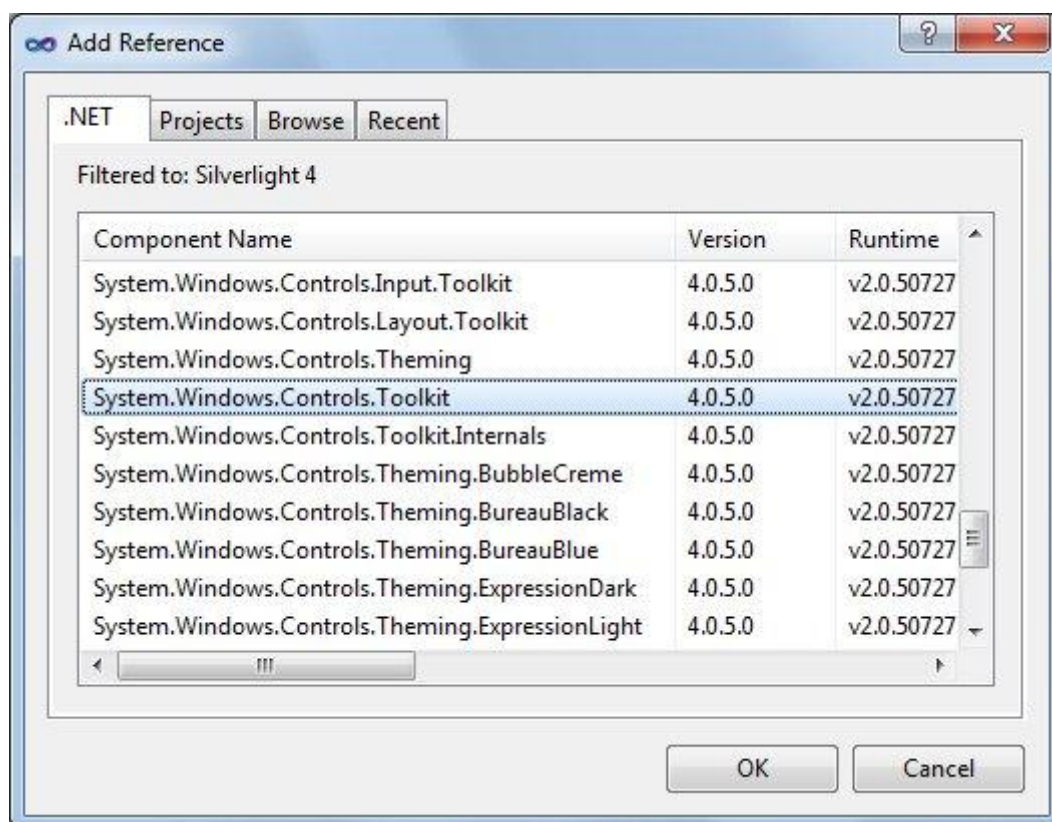
It's necessary to add report viewer component to the Silverlight application for the report display. In order to do that, you should add a reference to the PerpetuumSoft.Reporting.Silverlight.Client assembly, containing ReportViewer (right-click References of the SampleApplication in "Solution Explorer" and choose Add Reference item in the popup menu).



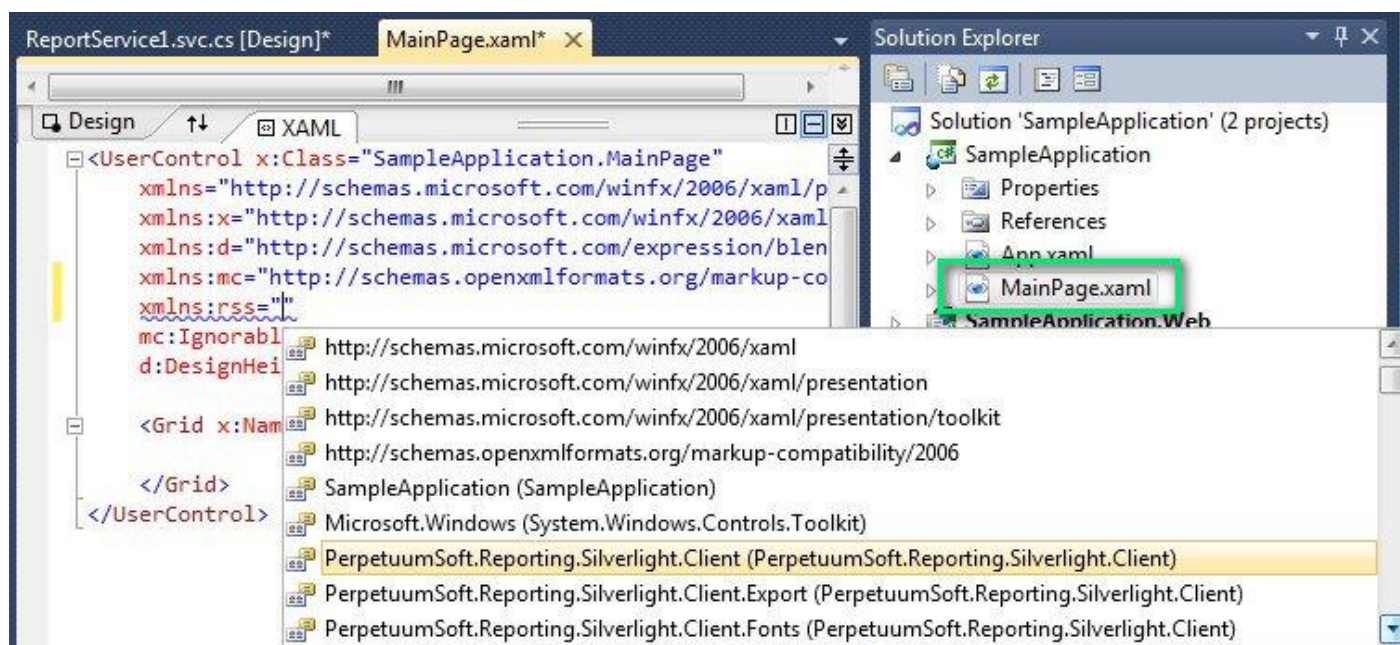




You should also add a reference to the System.Windows.Controls.Toolkit.dll assembly.



Open MainPage.xaml in the designer and add "rss" xml namespace for the PerpetuumSoft.Reporting.Silverlight.Client assembly.



Then add ReportViewer element to the Grid Section.

```
<rss:ReportViewer x:Name="ReportViewer1"
                  ServiceUrl="http://localhost:5555/ReportService.svc"
                  ReportName="CustomersReport"/>
```

After you made changes the UserControl section should look like this:

```
<UserControl x:Class="SilverlightApplication.MainPage"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:rss="clr-namespace:PerpetuumSoft.Reporting.Silverlight.Client;assembly=PerpetuumSoft.Reporting.Silverlight.Client"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
        <rss:ReportViewer x:Name="ReportViewer1" ReportName="CustomersReport"
                        ServiceUrl="http://localhost:5555/ReportService.svc"/>
    </Grid>
</UserControl>
```

Open MainPage source code and change/add the lines for code to look like this:

```
public partial class MainPage : UserControl
{
    public MainPage()
    {
        InitializeComponent();
        Loaded += new RoutedEventHandler(MainPage_Loaded);
    }

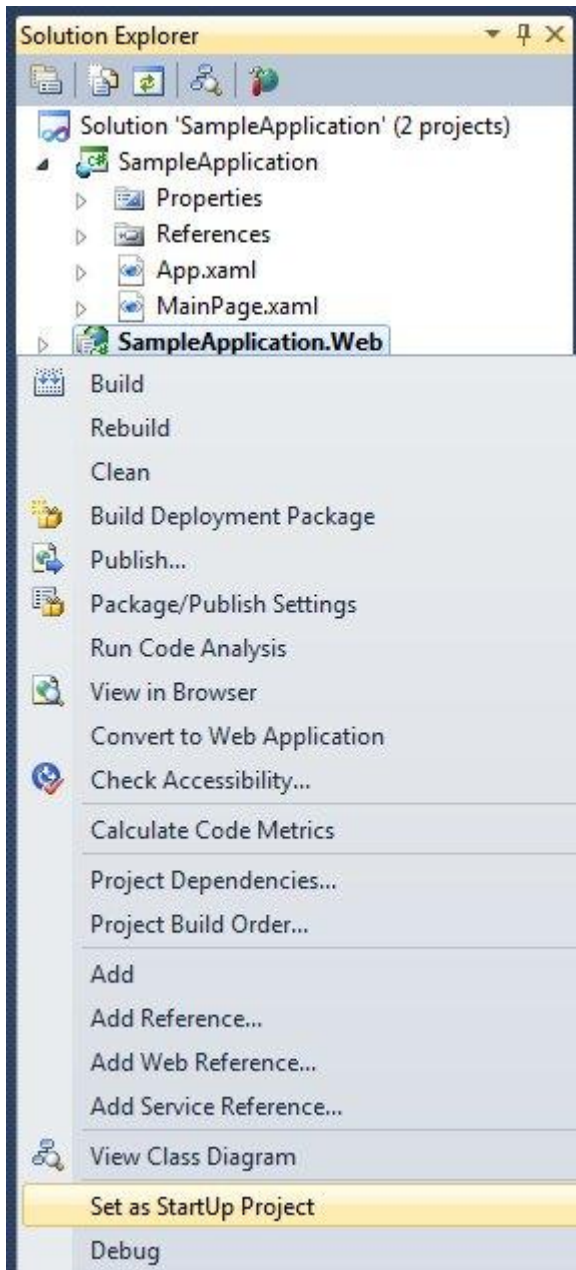
    void MainPage_Loaded(object sender, RoutedEventArgs e)
    {
        ReportViewer1.RenderDocument();
    }
}
```


}

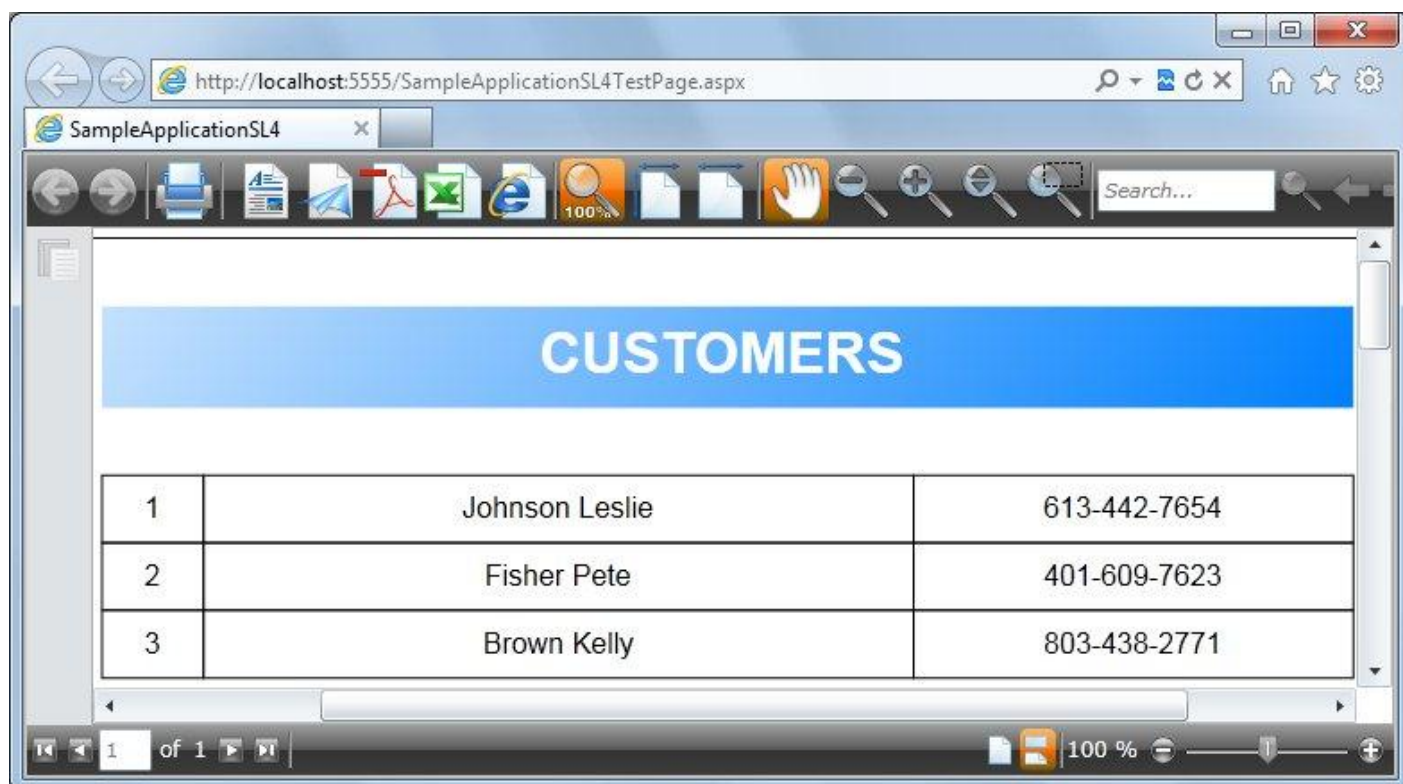
Note: RenderDocument() invocation leads to the rendering of the current report on the server and its displaying in the Report Viewer.

**Step6.** Launching application.

Set the SampleApplication.Web application as a StartUp. In order to do that, right-click SampleApplication.Web and choose Set as StartUp Project in the popup menu.



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



### Conclusion

We have examined basic steps and got a simple and quite operable application. We didn't have to write thousand lines of code – we only used ready-made implementation. It will be enough in the most cases. If required behavior may differ greatly from the one provided by default, since you can change not only many aspects of the SharpShooter Reports.Silverlight work but also the appearance of the report viewer.





## SharpShooter Gauges

### Destination and Basic Features

The SharpShooter Gauges is intended for creating and using indication and graphic user interface manipulation elements.

Nowadays most developers strive to create the most naturally looking, convenient and unique application interfaces. The SharpShooter Gauges is an ideal solution of this task.

Full-featured graphics editor allows creating new visual components with the unique look and functionality by means of only a few mouse clicks. It can be both common controls such as Progress Bars, Meters, Dials, Sliders, Gauges, Odometers, Thermometers, Switches etc. and specialized elements for your industry: Manipulators, Scales, Special-Purpose Devices and many others.

Each SharpShooter Gauges control consists of visual and non-visual objects interacting with each other. Due to these objects use, a developer can create any necessary visual control.

To assign separate elements' properties the expressions can be used, it helps to adjust the property depending on the current instrument status, as well as on the mouse status.

The use of the OPCChannel component gives a unique opportunity to incorporate real-time data channels with the OPC industry standard. The binding can be implemented both in read only and write modes.

A created instrument can be saved to a file for reuse.

### SharpShooter Gauges Concepts

The main concept of SharpShooter Gauges is the ability for quick and convenient creation of original control objects according to specified requirements without challenging a developer to have some special knowledge or high qualifications.

The SharpShooter Gauges contains components allowing to display the instrument and to manage it.

The Instrument is a compound object with hierarchical structure. In the instrument structure there are primitives intended for assigning the instrument appearance and behavior. The instrument allows designing some control object.

To create an instrument, the designer, that is also included in the SharpShooter Gauges delivery package, is used. The designer allows creating an instrument with the help of more simple elements, included in it in different combinations, and also to set the separate properties of the primitives, using convenient graphic means. To bind the primitives together and to assign the complex functionality, the expressions can be used.

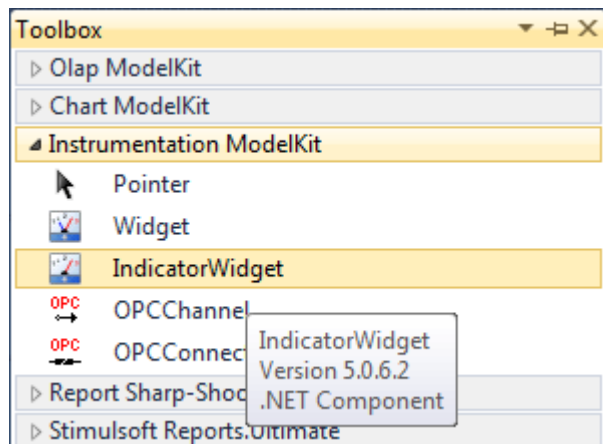
Different combinations of primitives, included in the instrument, the flexibility in settings properties of different elements, the use of expressions allow creating the control element, according to the developer's requirements.

## Getting Started

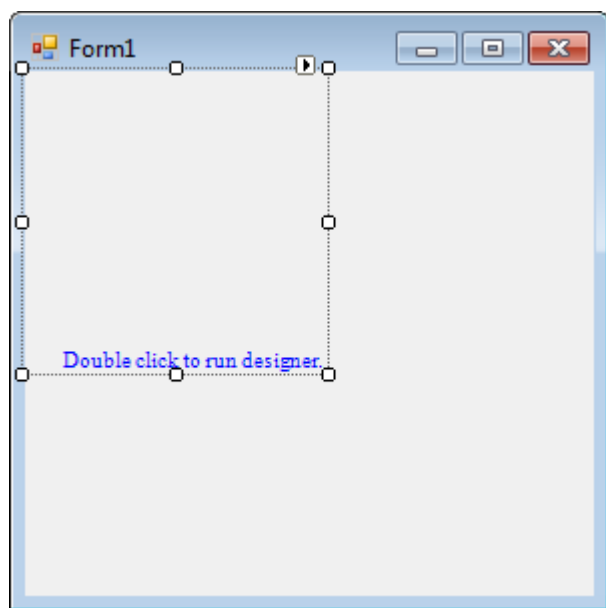
This article describes how a simple Windows Forms control element can be created by means of the SharpShooter Gauges.

Run Microsoft Visual Studio and create a new project – C# Windows Application.

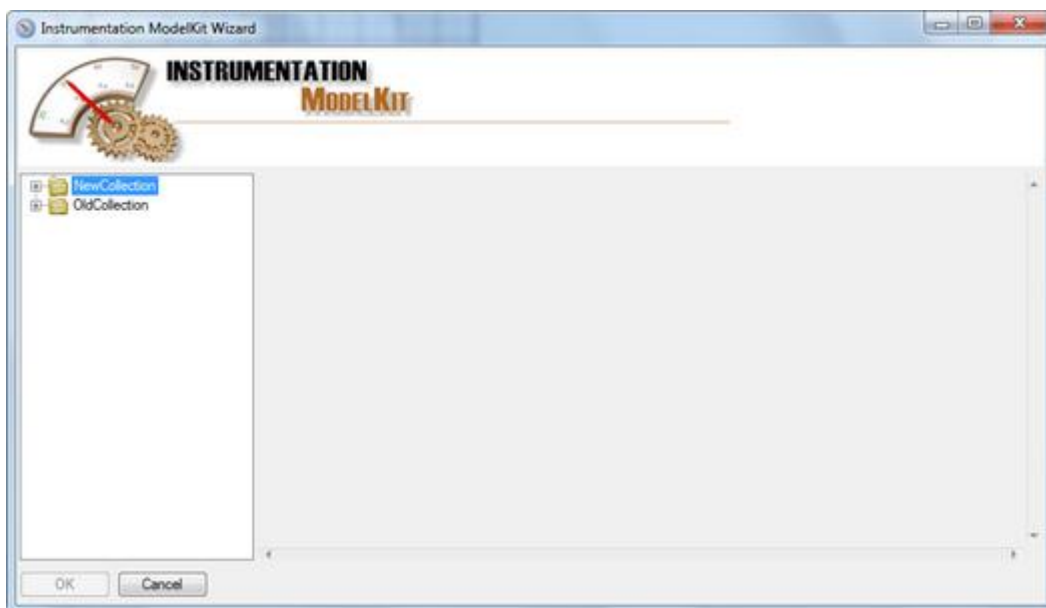
Put the **IndicatorWidget** component onto the form.



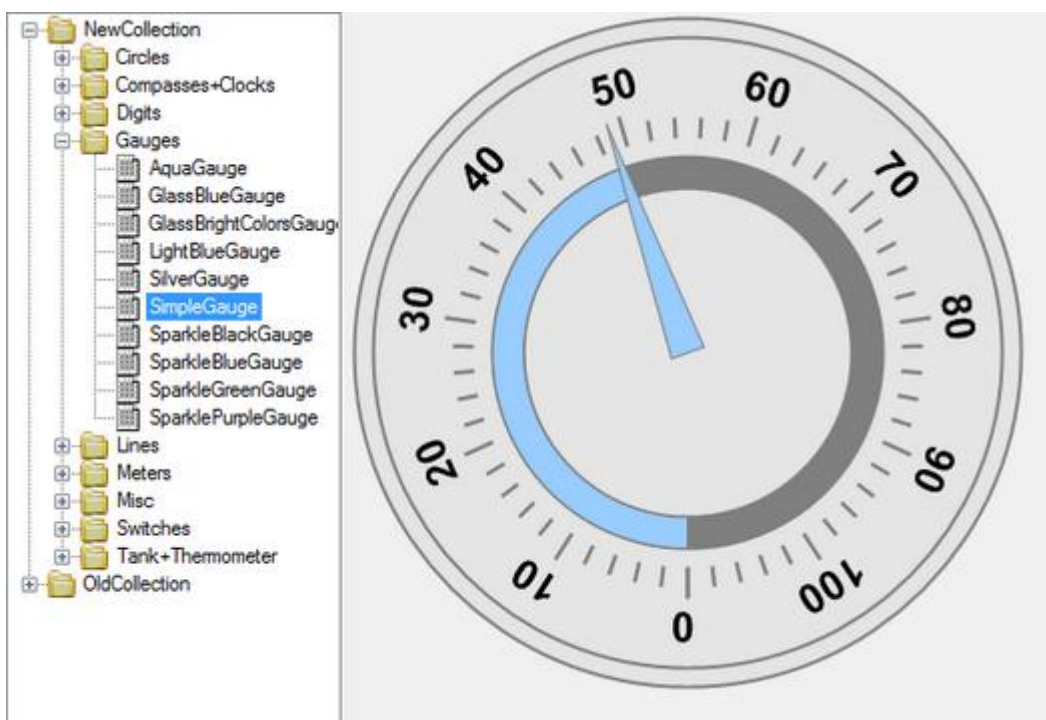
IndicatorWidget is a visual component intended for displaying an instrument.



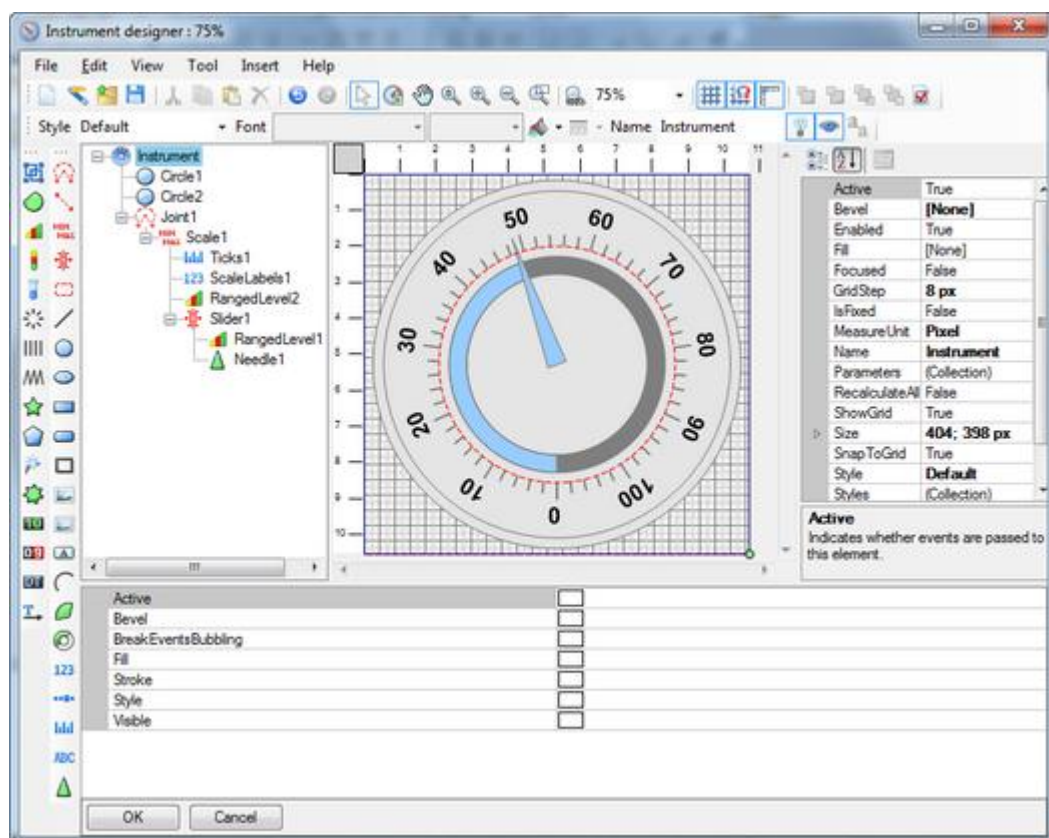
It is necessary to assign the instrument that will be displayed in this component. Double click to run the instrument designer. After the designer is launched, the dialog window in which the instrument template can be selected appears.



On the left of the window the instruments templates that can be used are presented. Choose the appropriate instrument template. For example, choose the SimpleGauge instrument from the Gauges group:



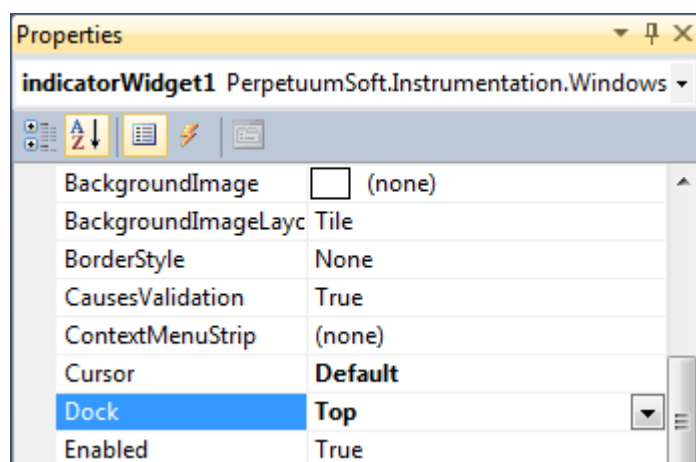
Click "OK", and the selected element will be added to the designer for editing.



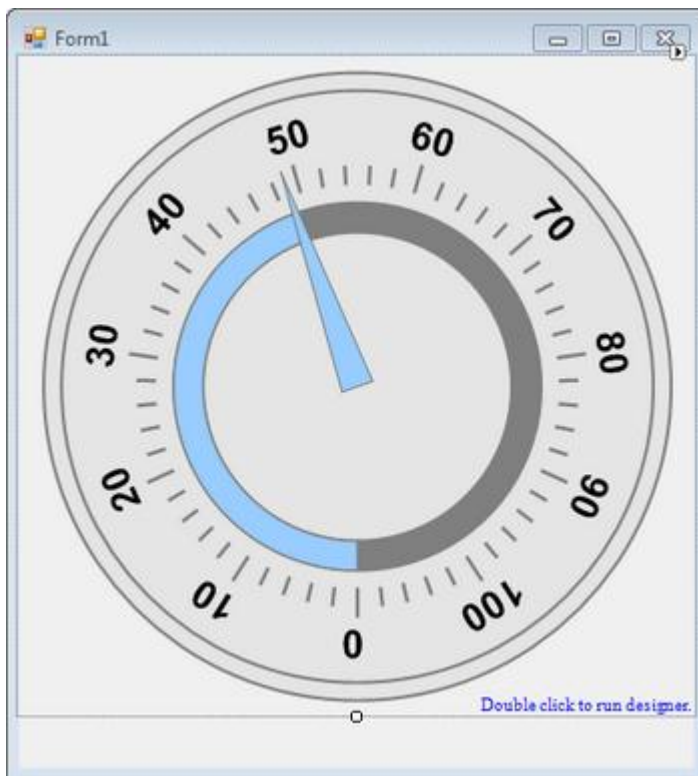
The instrument consists of the separate elements, each of which has its own functionality and properties. The designer allows adding, deleting and adjusting their properties. To manipulate the element it is necessary to select it with a mouse in the instrument window or in the tree, displaying the instrument structure.

Exit the designer, pressing the "OK" button for confirming changes. The final instrument will be used by the IndicatorWidget component.

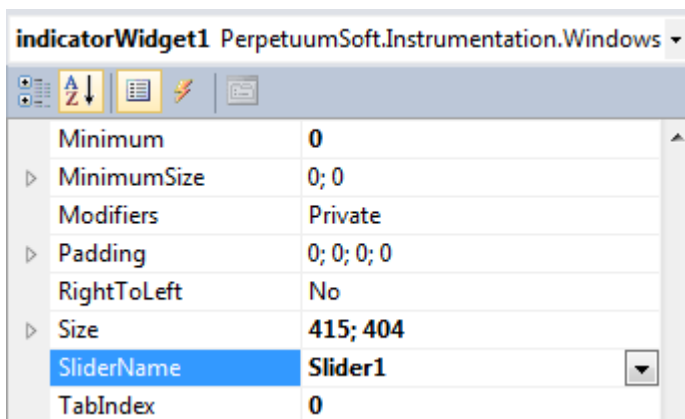
Set indicatorWidget1.Dock property to Top and change Form size.







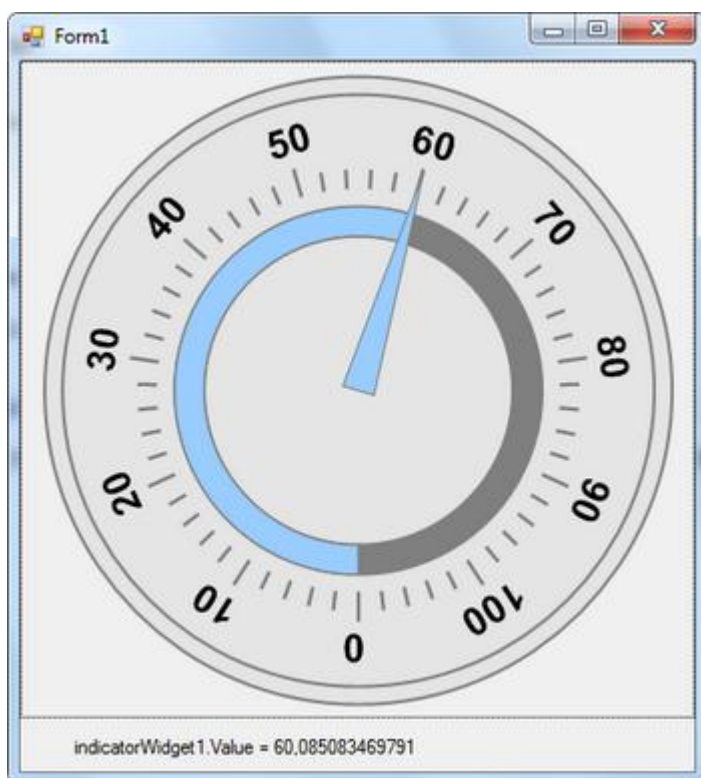
Specify the slider that will be used to receive the Value property to the component. To do it set the "Slider1" value to the *SliderName* property in the property grid.



Place the Label component onto the form for displaying the current IndicatorWidget value. Write the *ValueChanged* event handler for the indicatorWidget1.

```
private void indicatorWidget1_ValueChanged(object sender, EventArgs e)
{
    label1.Text = "indicatorWidget1.Value = " + indicatorWidget1.Value.ToString();
}
```

Run the application.



## Elements Description

### The General Instrument Model

The Instrument is a compound object with a hierarchical structure. Any elements can be included in the instrument. The elements can be structurally divided into two groups:

- simple;
- compound (elements that can include any other elements).

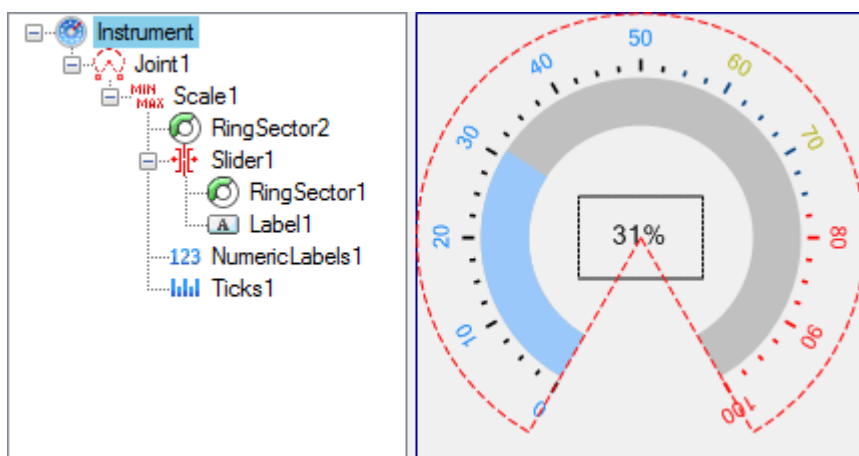
The elements can be functionally divided into:

- visual (are intended for instrument appearance assigning);
- non-visual (intended for elements' binding and grouping, as well as for assigning instrument functionality).

The elements can be included in the instrument in different combinations. However, for practical use the instruments, specifically intended for displaying and assigning some certain values, are utilized.

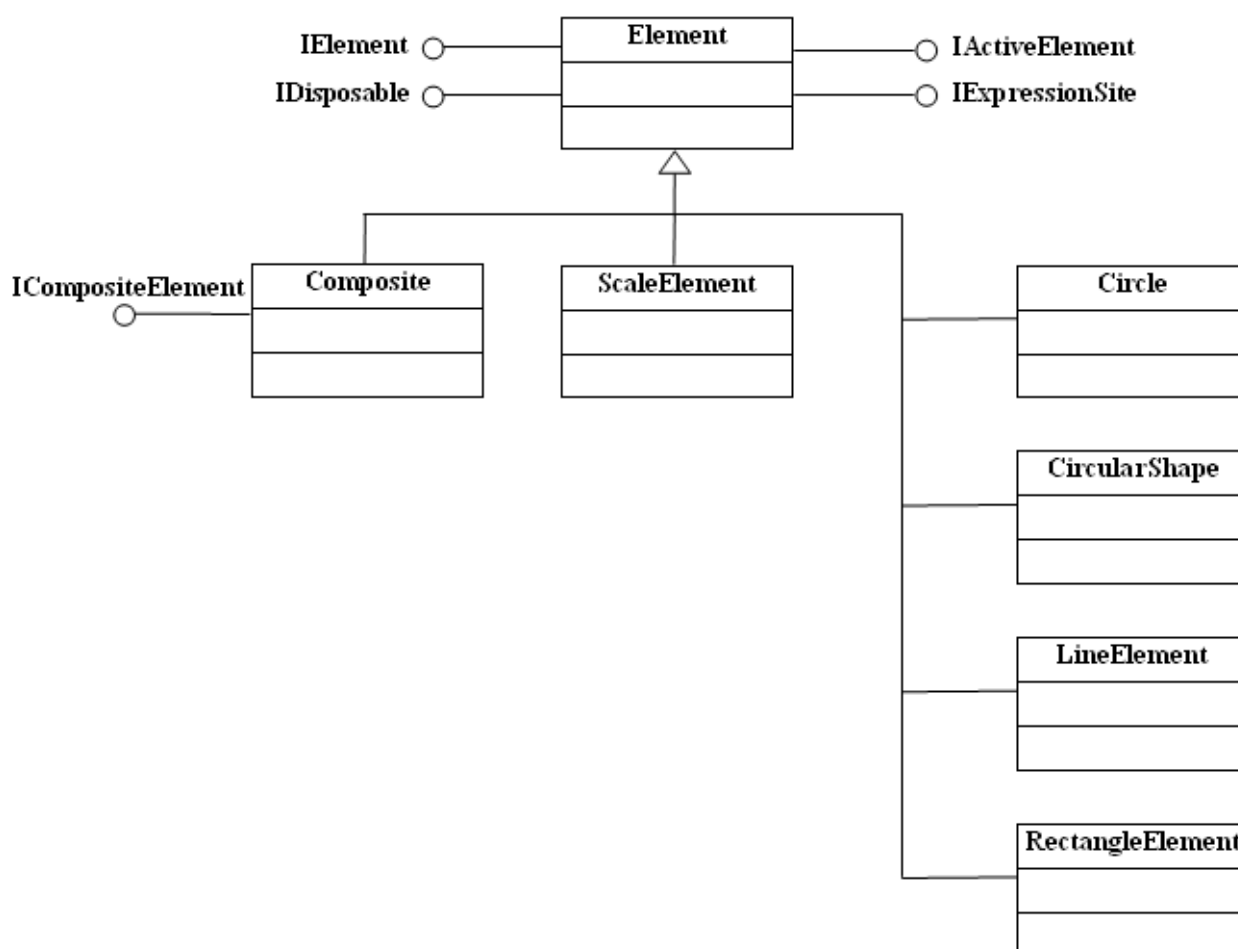
Such elements are built according to the following scheme. One or several scales assigning and displaying value range are included in the instrument. Several sliders displaying (assigning) the current value are included. The elements, displaying the value in some other form, for example in the form of text line, can also be included.

For example, to build a simple instrument the following structure is used:



### Elements Hierarchy

Let's consider the hierarchy of classes, used during the instrument creation.



The base class for all elements in the SharpShooter Gauges is the **Element** class; it is inherited from the System.Object and implements the PerpetuumSoft.Framework.Model.IElement, PerpetuumSoft.Framework.Model.Runtime.IActiveElement, PerpetuumSoft.Framework.Expressions.IExpressionSite and System.IDisposable interfaces.

The PerpetuumSoft.Framework.Model.IElement interface indicates that the Element object is the element of some document object model.

The PerpetuumSoft.Framework.Model.Runtime.IActiveElement interface indicates that the element can handle the mouse events.

The `PerpetuumSoft.Framework.Expressions.IExpressionSite` interface indicates that the element is the function and object source for expressions.

The `Element` class has the `Name` property, that assigns a name to the element, and properties used for drawing the element, such as `Style`, `Fill` and `Stroke`.

The `Element` class is the parent for the following classes: `Composite`, `ScaleElement`, `Circle`, `CircularShape`, `LineElement` and `RectangleElement`.

The **Circle** class represents the figure in the form of the disk.

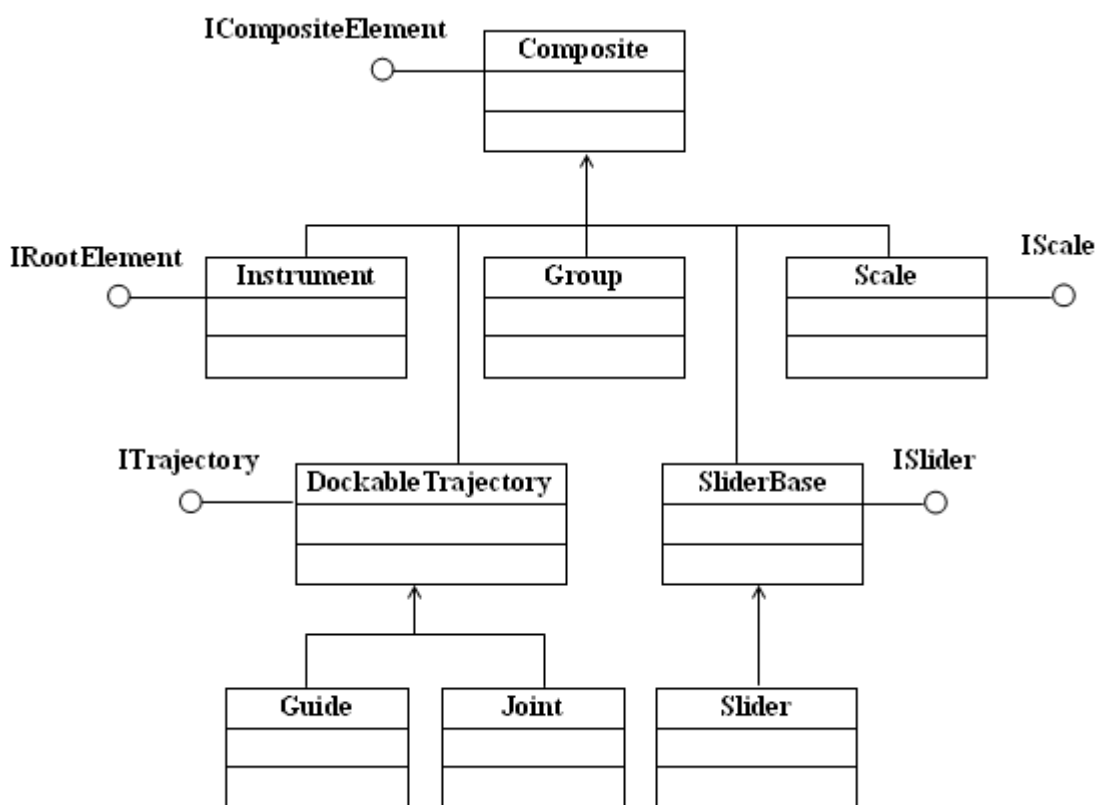
The **CircularShape** class represents the base class for all the figures containing center and radius.

The **LineElement** class is the base class for all the figures that have start and end points.

The **RectangleElement** class is the base class for all the figures containing center, width and height.

The **ScaleElement** class represents the base class for all objects placed on the scale.

The **Composite** class is the base class for all container-elements, i.e. elements capable of involving other elements. Let's consider it in details.



The `Composite` class realizes the `ICompositeElement` interface, indicating that this class is a container for the other elements and provides the access to these elements. `Instrument`, `Group`, `Scale`, `SliderBase`, and `DockableTrajectory` classes are inherited from the `Composite` class.

The object of the `Instrument` class represents the root element for all elements hierarchy and realizes the `IRootElement` interface indicating that it is the root element.

The object of the **Group** class is a non-visual element intended for grouping the other elements.

The object of the **Scale** class is a non-visual element assigning the scale range.



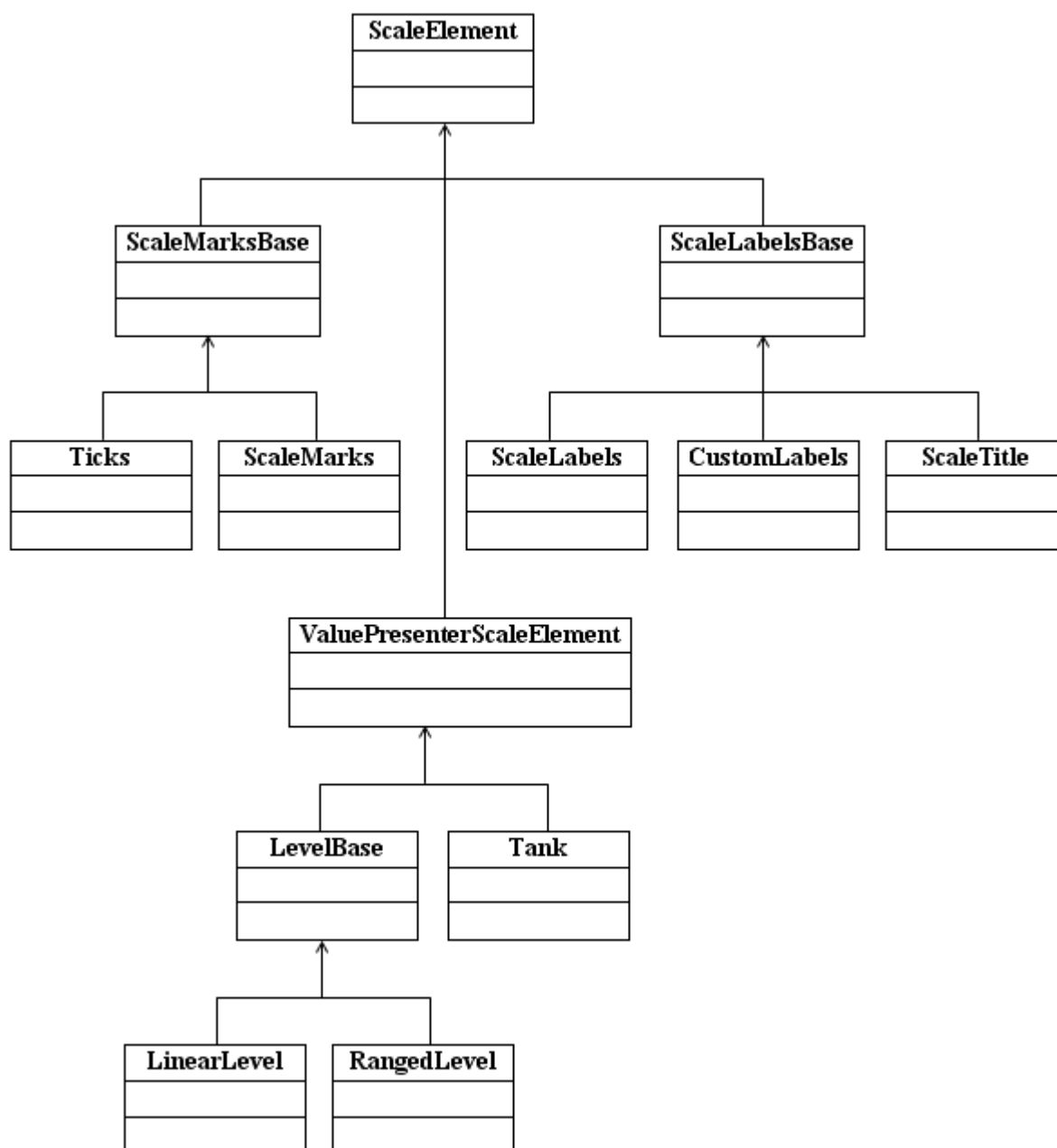
The **SliderBase** class represents the base class for sliders. A slider has the current value that can be changed with the help of a mouse. It is possible to define the point corresponding to the current Slider value located at some distance from the trajectory in which the Slider is included.

The **Slider** class, the which object is a non-visual element representing the current value, is inherited from the SliderBase class.

The **Guide** and **Joint** classes realize the ITrajectory interface indicating that classes represent trajectories. Guide is the trajectory having the form of the line. Joint is the trajectory having the form of the arc.

The abstract **DockableTrajectory** class realizes the ITrajectory interface indicating that the class represents Trjectories. Cuide is a linear trajectory. Joint is a circular trajectory.

Let's consider the **ScaleElement** class in details.



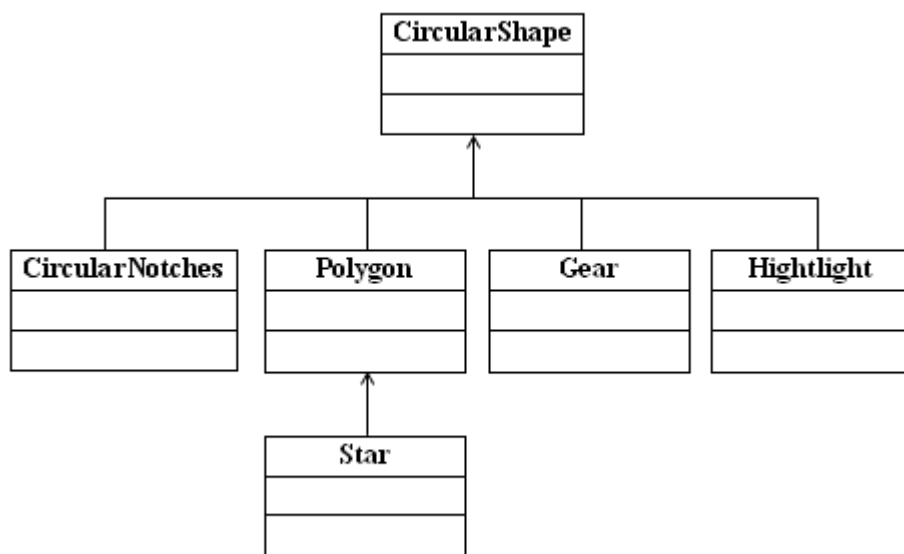
The descendants of the ScaleElement class are classes intended for scale visualization.

The **ScaleMarksBase** class represents marks on the scale. It is a base class for the Ticks and ScaleMarks classes. Objects of the Ticks class are the ticks on the scale. Objects of the ScaleMarks class are the marks on the scale.

The abstract **ScaleLabelsBase** class represents text labels on the scale. It is a base class for the ScaleLabels, CustomLabels, and ScaleTitleclasses. The ScaleLabels class represents numeric labels within the scale range. The CustomLabels class represents a collection of scale text labels. The ScaleTitle represents a scale title.

Descendants of the abstract **ValuePresenterScaleElement** are intended for displaying scale values. The Tank class represents linear scale value as tank level. The abstract LevelBase class is a base class for the LinearLevel and RangedLevel classes. LinearLevel visualizes linear scale value as a level. RangedLevel is intended for displaying scale value as level with variable width.

Let's consider the **CircularShape** class.



CircularNotches, Polygon, Gear, and Highlight classes are inherited from the CircularShape class.

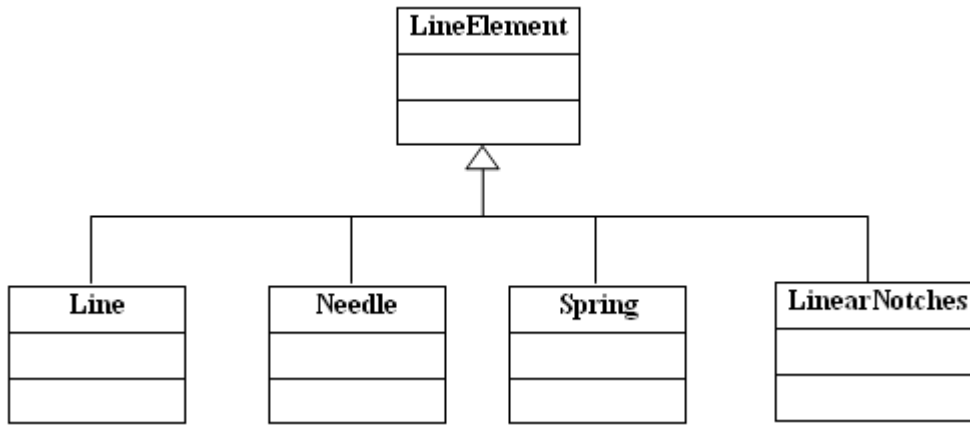
The object of the **CircularNotches** class creates the notch effect. It is used for the slider appearance design.

The **Polygon** class represents a polyhedron and has the Sides property that indicates the quantity of faces. The Star class, which represents a star, is inherited from it.

The **Gear** class represents gear.

The **Highlight** class is intended for highlight emulation.

Let's consider the **LineElement** class.



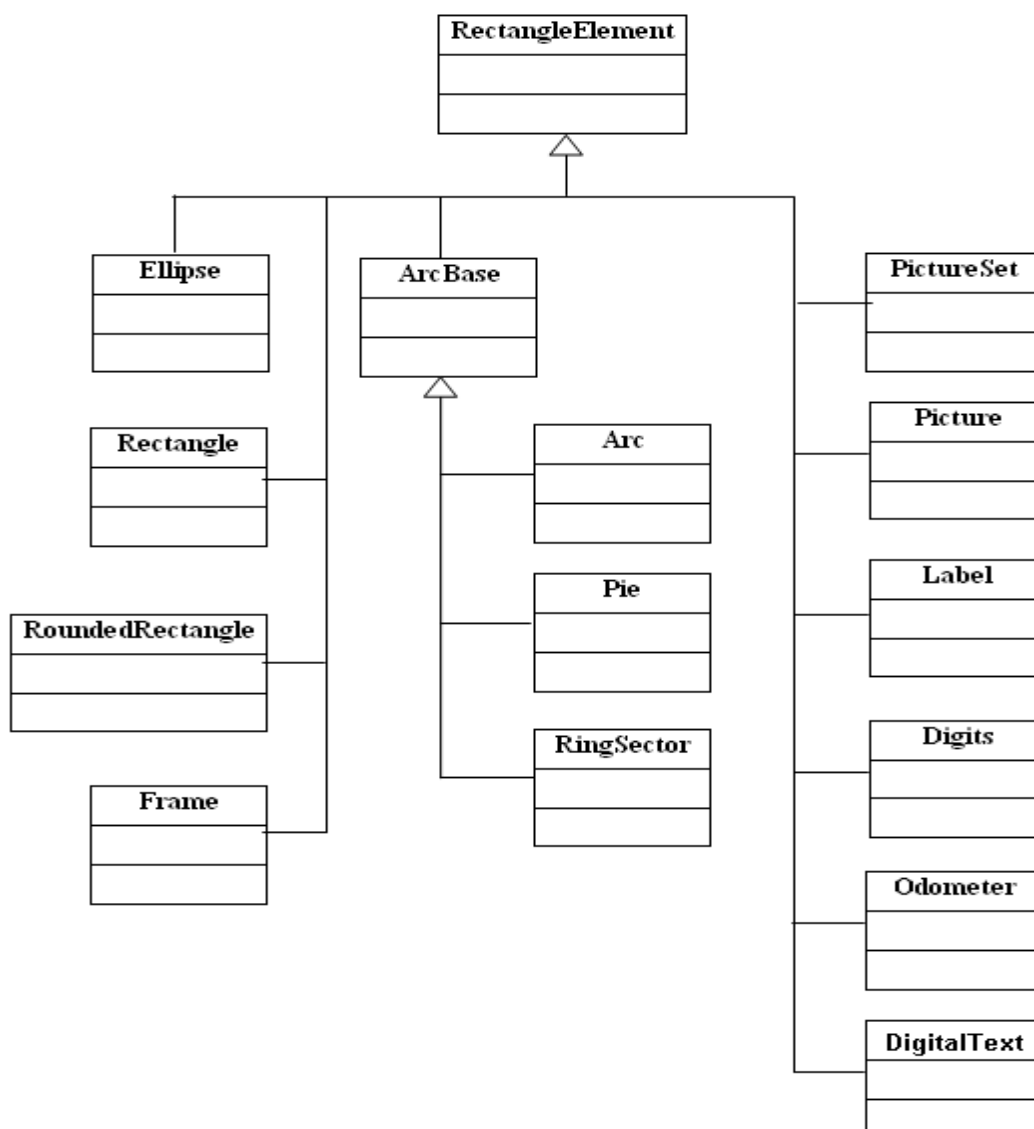
The **LineElement** class is the base class for Line, Needle, Spring and LinearNotches classes. The Line class represents a line.

The **Needle** class represents a pointer, the form of which is assigned with the help of the NeedlePoints property. It is used for slider visualization.

The **Spring** class represents a spring.

The object of the **LinearNotches** class creates the effect of linear notches. It is used for the slider appearance design.

Let's consider the **RectangleElement** class.



Ellipse, Rectangle, RoundedRectangle, Frame, ArcBase, Picture, PictureSet, Label, Digits and Odometer are inherited from the RectangleElement class.

The **Ellipse** class represents a figure in the form of ellipse.

The **Rectangle** class represents a figure in the form of rectangle.

The **RoundedRectangle** class represents a figure in the form of rectangle with the rounded angles.

The **Frame** class represents a rectangular frame.

The **ArcBase** class is the base of the Arc, Pie and RingSector classes that accordingly represent an ellipse arc, an ellipse sector and an elliptical ring sector.

The objects of the **Picture** class are intended for displaying the picture, assigned in the Image property.

The objects of the **PictureSet** class are intended for displaying one picture from the assigned pictures collection.

The objects of the **Label** class are intended for inscription displaying.

The objects of the **Digits** class are intended for digital indicator emulation.





The objects of the **Odometer** class are intended for mechanical meter emulation.

### Description of General Properties

Let's consider the Instrument class properties, peculiar to all descendants' elements in details.

The *Name* property assigns the unique name within the entire instrument.

The *Fill* property assigns the fill, which will be used for element drawing.

Line tracing style, which will be used for element drawing, is assigned in the *Stroke* property.

The *Style* property assigns the style name of the element, used for element drawing. Available styles are assigned in the Styles property of the instrument, the element belongs to. If the element has no Fill and/or Stroke properties assigned, the style with the name assigned in the Style property will be applied.

The *Smooth* property assigns the quality of output graphic.

The *Visible* property assigns element's visibility.

The *Active* property indicates whether an element will perceive the events from the mouse.

The *BreakEventBubbling* property indicates whether the events from the mouse will be passed to the parent element for the further processing.

The *RecalculateAll* property indicates for what elements the expressions will be recalculated, if the element state was changed. If RecalculateAll is set to false, the expressions will be recalculated only for the selected element and all the elements included in it, otherwise the expressions of all instrument elements will be recalculated.

The process of receiving events from a mouse and their handling is produced in the following way:

The visual element receives events from the mouse when its pointer is over the element. The element handles these events and, if the BreakEventBubbling property is set to false, passes the event to the parent for handling. Thus, the events received from the mouse are spread from the visual element up the instrument tree. Non-visual elements can receive the event from the mouse only from the included visual elements.

The *Size* property is the size of the instrument work area.

The *Enabled* property indicates whether the instrument will react on the user actions.

The *Focused* property indicates whether the instrument is in the focus.

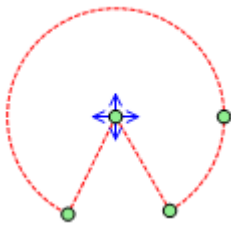
During the instrument development these values can be set arbitrarily. When the instrument is displayed in Widget, these properties are set according to Enabled and Focused Widget properties. These properties are intended to implement the instrument appearance and behavior depending on the state of the Widget, in which the instrument is displayed.

The following properties are used only in the instrument design mode and saved with the instrument: *SnapToGrid* – snap to grid, the *ShowGrid* property - the grid visibility, the *GridStep* property – the step of the grid, the *MeasureUnit* property – the measurement units used.

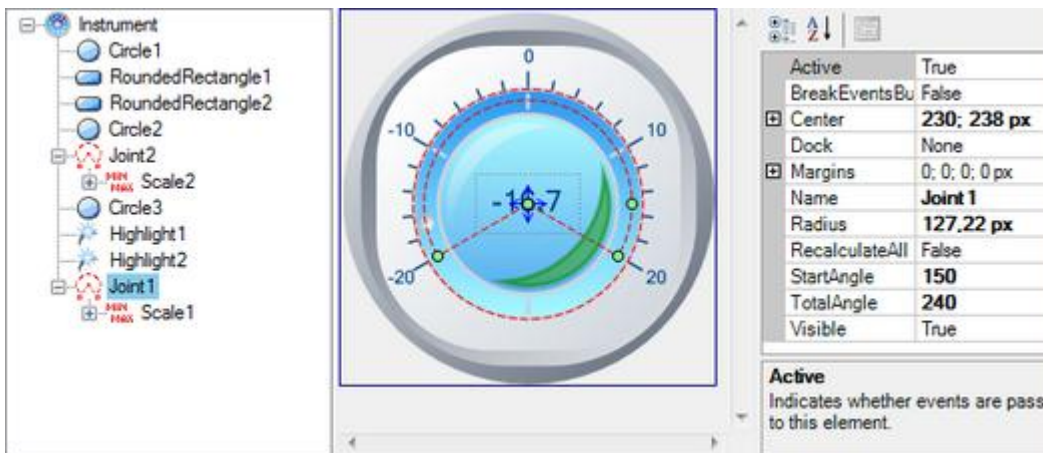


### The Description All Elements and Their Properties

The **Joint** element is the trajectory in the line of an arc.



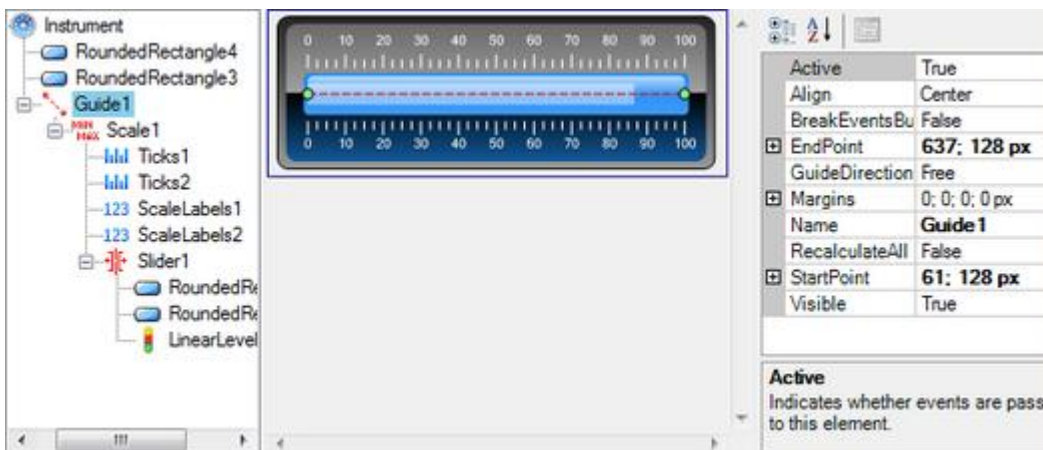
The arc is assigned by the center (the *Center* property), by the radius (the *Radius* property), by the starting angle (the *StartAngle* property) and by the total angle (the *TotalAngle* property). The angles are counted clockwise, from the horizontal axis. The Joint element is used for dials creation.



The **Guide** element is the trajectory along the line.



The line is assigned by the start (the *StartPoint* property) and the end (the *EndPoint* property) points. This element is used for linear scales creation.



The **Scale** element is a non-visual element that assigns the range of scale value changes. The range is assigned with the help of *Minimum* and *Maximum* properties. For the scale visualization such elements as Ticks, NumericLabels and TextLabels are included into the Scale element.

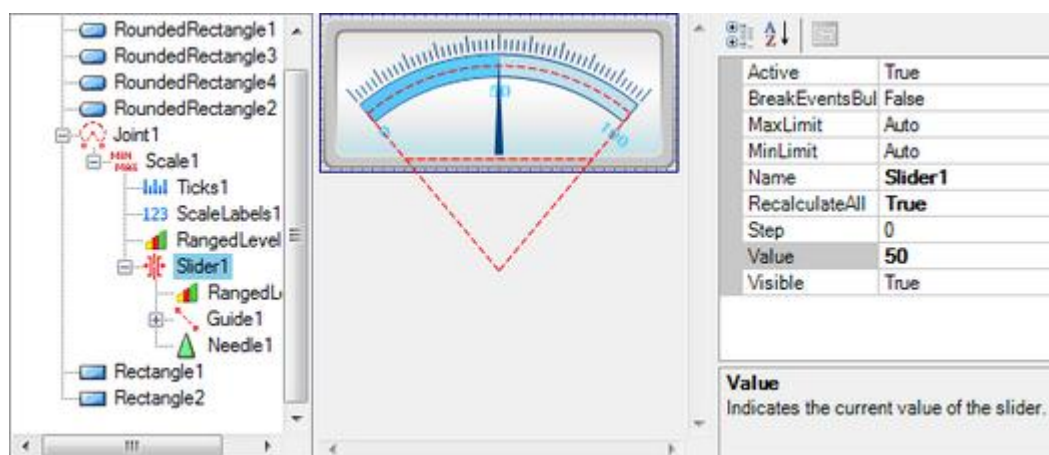


The **Slider** element is a non-visual element that assigns the current value (the *Value* property) on the scale, in which this slider is included. The Slider handles the events from the mouse in the following way: it synchronizes the *Value* property according to the current position of the mouse pointer and the geometry of the trajectory, in which it is included. As the Slider element is a non-visual one, it cannot directly receive the events from the mouse. The events from the mouse are passed to the Slider by the visual elements included in it. Thus, in order to make the Slider react on the mouse moves, it is necessary to include some visual element in it.

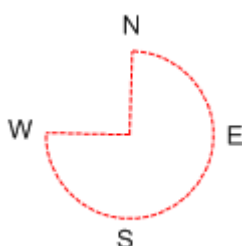
The *MaxLimit* and *MinLimit* assign range of changing value and can process both absolute and relative values. The *UseLimit* property defines, if the assigned range of changing value is used.

Various visual elements, included in the Slider and bound to it by expressions, can be used for visualizing current Slider value.

The group element is a non-visual element used for grouping other elements.

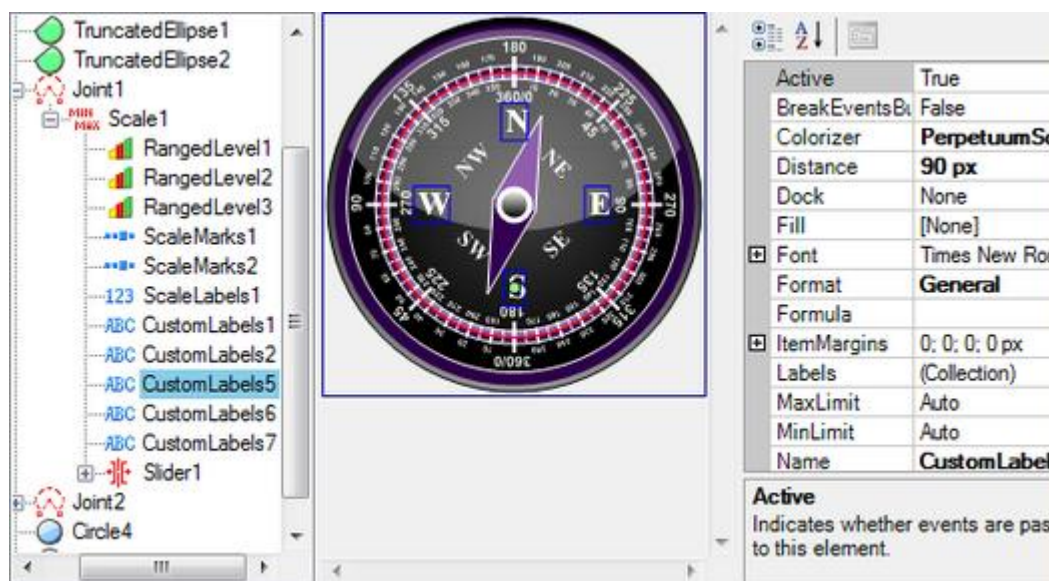


The **CustomLabels** element represents text labels on the scale, that are not bound to scale values.

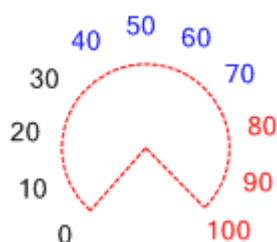


Text labels as well as corresponding scale values are assigned in the Labels collection. Scale value can be assigned both by absolute value and relative value (in percentage from the scale range). Labels, which value is set to Auto, will be evenly distributed along the scale subject to benchmark assigned by the *Origin* property. Shift from the Scale is assigned in the *Distance* property.

The *Dock* property defines relative position of the CustomLabels element subject to shifts assigned in the *Padding* property. The *OddLabelDistance* property sets shift of odd labels relative to even ones. Labels alignment relative to each other is assigned by the *Position* property. The *TextAlignment* property defines marks alignment relative to the corresponding scale value. Text angle is assigned in the *Angle* property, and text orientation is set in the *TextRotationMode* property. The *ItemMargins* property defines text shift from the label area margins. The *ShowSuperposableLabels* defines if the superposable labels will be displayed. If you need to display labels only in the specified range, use the *MaxLimit* and *MinLimit* properties assigning range bounds. The *UseLimit* property indicates that limitation of element output will be executed. This element should be included in the scale.



The **ScaleLabels** element represents numeric labels on the scale.

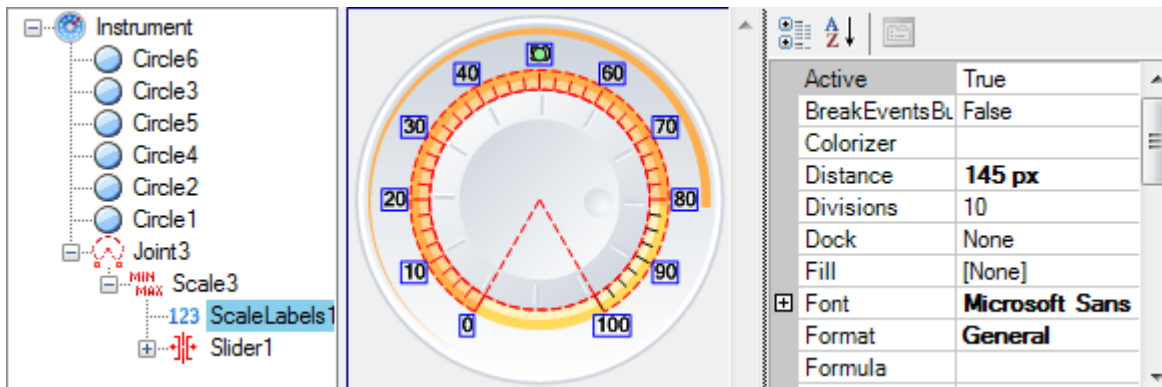


Values are displayed within the range, set in the scale, and step, defined by the *Step* property. If the *Step* property is set to Auto, divisions' amount is set in the *Divisions* property. The origin property defines a benchmark. If the *UseRoundValues* is set to true, scale marks will be distributed along the scale in the way that the corresponding values contain minimal amount of fractional numbers; the number of divisions will be less or equal to the *Divisions* property value. When the element is drawn, these labels will be evenly distributed along the scale. Shift from the scale is assigned by the *Distance* property.

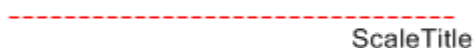
The *Dock* property defines relative position of the ScaleLabels subject to shifts, assigned in the *Padding* property. The *OddLabelDistance* property sets shift of the odd labels relative to even ones. Labels alignment relative to each other is set in the *Position* property. The *TextAlignment* property sets marks alignment relative to the corresponding scale value. Label angle is set in the *Angle* property, and label orientation is set in the *TextRotationMode* property. The *ShowSuperposableLabels* defines if the superposable labels will be displayed. The *ItemsMargins*



property defines text shift from the Label area bounds. If you need to display Labels only within the assigned range, use the *MaxLimit* and *MinLimit* properties that set range bounds. The *UseLimit* property indicates that limitation of the element output will be executed. This element should be included in the scale.

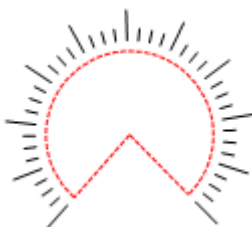


The **ScaleTitle** element represents the scale title.

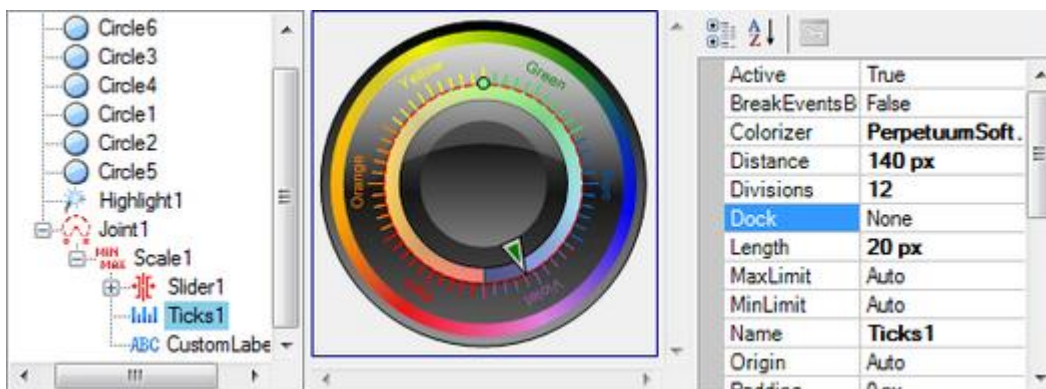


Title text is defined in the *Text* property. Element position is assigned is set in the *Origin* property. The *Dock* property assigns relative position of the Scaletitle subject to shifts set in the *Padding* property. The *TextAlignment* defines title alignment. Text angle is set in the *Angle* property, and text orientation is assigned in the *TextRotationMode* property.

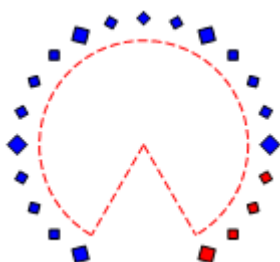
The **Ticks** element is the ticks on the scale.



Values are displayed within the range, assigned in the scale and step defined in the *Step* property. The number of the ticks is assigned in the *Divisions* property. The *Origin* property specifies the bench mark. If the *UseRoundValues* property is set to true, scale ticks will be distributed in the way that the corresponding values contain minimal amount of the fractional numbers; the number of divisions will be less or equal to the *Divisions* property value. The length of the tick is assigned in the *Length* property. The intermediate ticks, which divide this range according to the *SubDivisions* property, are output between the ticks in the range. The length of the intermediate ticks is assigned in the *SubLength* property. The positional relationship of the ticks and the intermediate ticks can be assigned with the help of the *SubTicksPosition* property. When drawing the element, the ticks will be evenly spread in the line of the whole scale. The indent from the scale is assigned with the help of the *Distance* property. The *Dock* property determines the Ticks element position subject to shifts, set in the *Padding* property. If there is a necessity to output the ticks only in the assigned range, use *MaxLimit* and *MinLimit* properties that assign the range borders. The *UseLimits* property indicates that the limitation on element outlet will be used. This element is included in the scale.



The **ScaleMarks** element represents graphic markers on the scale.

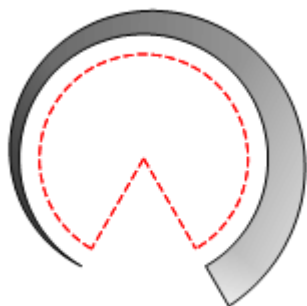


Shape type is assigned in the *Shape* property. Values are displayed within the range, set in the scale, and step, assigned in the *Step* property. If the *Step* property value is set to *Auto*, division amount is set in the *Divisions* property. The *Origin* property sets the benchmark. Marks size is assigned in the *MarkSize* property. *Submarks* are displayed in the within the range between marks, and they divide this range according to the *SubDivisions* property. Submarks size is assigned in the *SubMarkSize* property.

Marks and submarks relative position is set in the *SubTicksPosition* property. When the element is drawn, marks will be evenly distributed along the scale. Shift from the scale is set in the *Distance* property. The *Dock* property determines relative position of the marks subject to shifts, set in the *Padding* property. Marks angle is assigned in the *MarksAngle* property. If you need to display marks only in the specified range, use the *MaxLimit* and *MinLimit* properties, which assign range bounds. The *UseLimit* property indicates that limitation of the element output will be performed. The *ScaleMarks* element should be included in the scale.

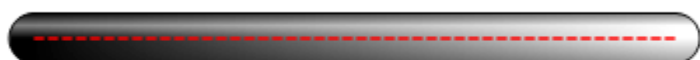


The **RangedLevel** element is intended for displaying intervals on the scale.



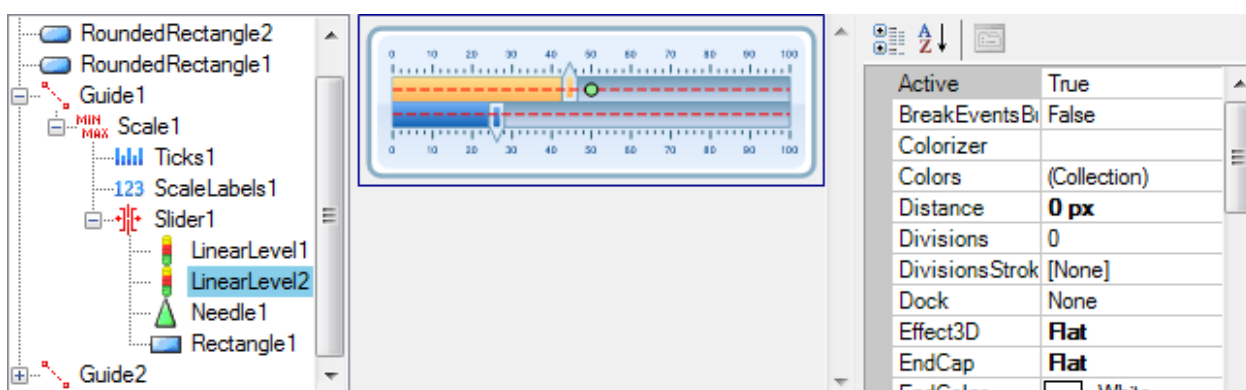
The *Value* property assigns value the element will be displayed up to. The *StartWidth* and *EndWidth* property set width of the element at the beginning and at the end of the scale. If the *Fill* property is not set, the *RangedLevel* element will be painted with colors assigned in the *Colors* property. If the *Colors* property is not set, the element will be painted with colors set by *StartColor* and *EndColor*. The *Divisions* property determines divisions amount. The *DivisionsStroke* property assigns type of lines used to divide divisions. The *AlignmentMode* property assigns the way element will bound to the scale. If you need to display markers only within the specified range, use the *MaxLimit* and *MinLimit* properties that set range bounds.

The **LinearLevel** element is intended for displaying intervals on the linear scale.

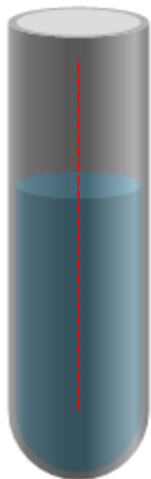


The *Value* property sets value the element will be displayed up to. The *Divisions* property determines division's amount. The *DivisionsStroke* property assigns type of lines used to divide divisions. If the *Fill* property is not set, the *RangedLevel* element will be painted with colors assigned in the *Colors* property. If the *Colors* property is not set, the element will be painted with colors set by *StartColor* and *EndColor*.

The *Width* property determines element width. The *StartCap* and *EndCap* properties assign style of displaying element-start and element-end. Style of displaying the whole element is determined in the *Effect3D* property. The *ShowAsThermometer* property determines if the element will be displayed as thermometer, i.e. there will be a circle with radius, set in the *PocketRadius* property, at the beginning of the scale. The *Dock* property determines type the of element's docking to the scale.



The **Tank** element is a visual element, intended for emulating a tank, filled with liquid. The element is assigned by the *Width* property, determining tank width, the *Depth* property, assigning tank depth, and the *TankWidth*, determining width of the tank walls. The *TankColor* and *LiquidColor* set tank color and liquid color correspondingly. Style of displaying the whole element is determined by the *Effect3D* property. The *Dock* property determines type the of element's docking to the scale.



The **Circle** element is a visual element representing the figure in the form of a circle.



The circle is assigned with the help of the center (the *Center* property) and radius (the *Radius* property).

The **Ellipse** element is a visual element representing the figure in the form of an ellipse.



It is assigned with the help of the center (the *Center* property) and size (the *Size* property). The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **Rectangle** element is a visual element representing the figure in the form of a rectangle.



It is assigned with the help of the center (the *Center* property) and size (the *Size* property). The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **RoundedRectangle** element is a visual element representing the figure in the form of a rectangle with the rounded angles.





It is assigned with the help of the center (the *Center* property) and size (the *Size* property). The smoothing radius is assigned in the *Radius* property. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **Arc** element is a visual element representing the figure in the form of an elliptical arc.

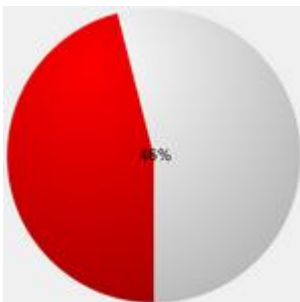


An ellipse is assigned with the help of the center (the *Center* property) and size (the *Size* property). The Arc element is assigned by the starting angle (the *StartAngle* property) and the summarized angle (the *SweepAngle* property). The *Angle* property assigns the figure rotation clockwise relatively to the center.

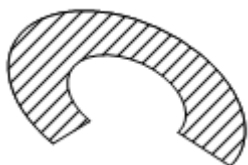
The **Pie** element is a visual element representing the figure in the form of an ellipse sector.



An ellipse is assigned with the help of the center (the *Center* property) and size (the *Size* property). A sector is assigned by the starting angle (the *StartAngle* property) and the summarized angle (*SweepAngle* property). The *Angle* property assigns the figure rotation clockwise relatively to the center.



The **RingSector** element is a visual element representing the figure in the form of an ellipse ring sector.



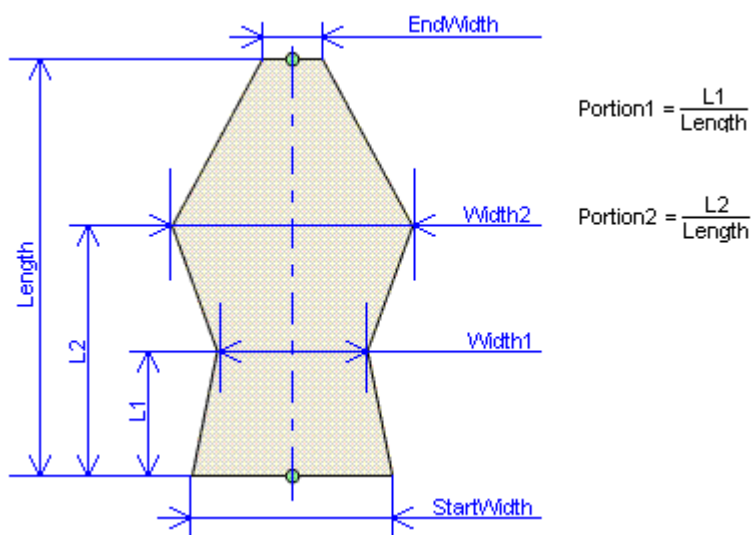
An ellipse is assigned with the help of the center (the *Center* property) and size (the *Size* property). A sector is assigned by the starting angles (the *StartAngle* property) and the summarized angle (the *SweepAngle* property). The ratio of the internal radius to the outer one is assigned in the *InternalRadius* property. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **Line** element is a visual element representing the figure in the form of a line.

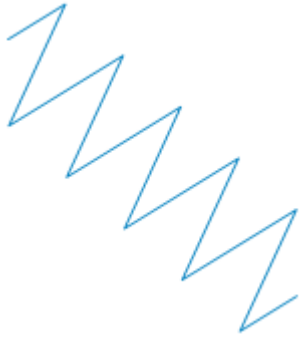


A line is assigned with the help of the *StartPoint* property and the *EndPoint* property.

The **Needle** element is a visual element representing the figure in the form of a needle. This element is intended for the Slider current value visualization. The *StartPoint* property assigns the start point of the needle. The *EndPoint* property assigns the end point of the needle. The needle base width is assigned in the *StartWidth* property and the needle tip width is assigned in the *EndWidth* property. To assign the complex geometry of the needle the *NeedlePoints* property is used. *NeedlePoints* is the collection of intermediate needle points descriptors. Each intermediate point descriptor is assigned by the needle width at a given point and the ratio between the distance from the needle base to a given point and the needle length. (See the image below). The *ShowMode* property assigns the needle displaying mode.



The **Spring** element is a visual element representing the figure in the form of a spring.



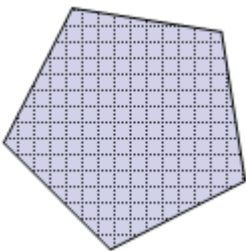
A spring is assigned by the starting point (the *StartPoint* property). Spring width is assigned by the *Amplitude* property. The number of coils is assigned by the *CoilCount* property.

The **Star** element is a visual element representing the figure in the form of a star.



The star position is assigned by the center (the *Center* property), the radius of a circle circumscribed round it (the *Radius* property) and the radius of an inscribed circle (the *InternalRadius* property). The number of star sides is assigned by the *Sides* property. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **Polygon** element is a visual element representing a regular polygon inscribed in the circle with the center assigned by the *Center* property.



A polygon size is assigned by the radius of a circle circumscribed round it (the *Radius* property). The number of polygon faces is assigned by the *Side* property. The *Angle* property assigns the figure rotation clockwise relatively to the center

The **Label** element is the visual element intended for text output.



The *Text* property assigns the output text. The rectangle, in which the text is output, is assigned by the center (the *Center* property) and by size (the *Size* property). Text alignment relatively to the rectangle is assigned by the *TextAlign* property. The *Font* property assigns the text font. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **Frame** element is a visual element in the form of a frame with the bevel.



The frame coordinates are assigned by the center (the *Center* property) and size (the *Size* property). The bevel style is assigned in the *BevelStyle* property. *DarkColor*, *LightColor*, *OuterColor*, *InnerColor* properties assign the colors used for drawing the bevel. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **Picture** element is intended for the picture output.



The picture is assigned in the *Image* property. Picture position and size are assigned by the *Center* and *Size* properties. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **PictureSet** element is intended for the output of one picture from the assigned pictures set. Picture collection is assigned in the image property. The number of output picture is assigned by the *ImageIndex* property. Picture position and size are assigned by the *Center* and *Size* properties. The *Angle* property assigns the figure rotation clockwise relatively to the center.



The **CircularNotches** element creates the radial notches effect.



It is used for control elements design. The notches are inside the circle assigned by the center (the *Center* property) and the radius (the *Radius* property). The number of the notches is assigned by the *Count* property, their length is assigned by the *Length* property. Colors, assigned by *DarkColor* and *LightColor* properties, are used for drawing the notches. The *Angle* property assigns the figure rotation clockwise relatively to the center.

The **LinearNotches** element creates the linear notches effect. It is used for control elements design.



The notches are placed parallel to the line assigned by the start and end points (the *StartPoint* and *EndPoint* properties). The distance between the extreme notches is assigned by the *Width* property. The number of the notches is assigned by the *Count* property. Colors, assigned by *DarkColor* and *LightColor* properties, are used for drawing the notches.

The **Digits** element is intended for digital indicator emulation.



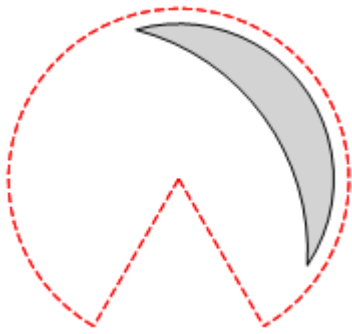
It is used for digital values output. The text, displayed in the indicator, is assigned in the *Text* property. The following symbols are allowed: numbers (0-9), "point", "dash", "colon", "space". The position and size are assigned by the *Center* and *Size* properties. The *Angle* property assigns the figure rotation clockwise relatively to the center. The indicator number is displayed by the figures, consisting of segments. Segments thickness is assigned by the *SegmentThickness* property. The interval between the segments is assigned by the *SegmentSpace* property. The Symbol width is assigned by the *DigitWidth* property, height is assigned by *DigitHeight* property, the space between the figures is assigned by the *DigitSpace* property. The color of figures active segments is assigned by the *ActiveColor* property, the color of inactive segments is assigned by the *InactiveColor* property.

The **Odometer** element is intended for mechanical meter emulation.



It is used for value output. The output number is assigned by the *Value* property. The position and size are assigned by the *Center* and *Size* properties. The *Angle* property assigns the figure rotation clockwise relatively to the center. The font of displayed figures is assigned by the *Font* property. The assigned value is output on different meter disks. The disk color, in which the number last rank is displayed, is assigned by the *FirstDigitBackFill* property; the figure color, output on this disk, is assigned by the *FirstDigitForeFill* property. The color of the other disks is assigned by the *BackFill* property; the color of the symbols, output on them, is assigned by the *ForeFill* property.

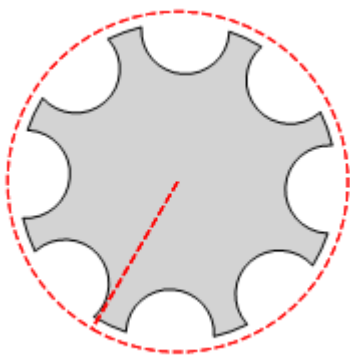
The **Highlight** element is intended for emulating highlights on the circular instruments.



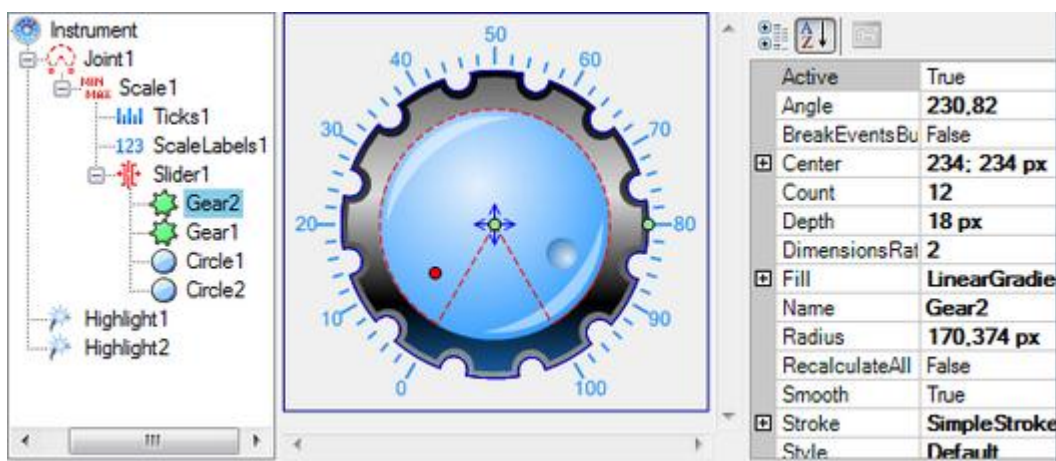
Element's position is set in the *Center* and *Radius* properties assigning center coordinates and radius. The *Angle* and *SweepAngle* properties set element's start angle and its angle of turn.

Active	True
Angle	165,96
BreakEventsBu	False
Center	208; 208 px
Fill	SolidFill
K	0,6
Name	Highlight 1
Radius	132 px
RecalculateAll	False
Smooth	True
Stroke	[None]
Style	Highlight 1
SweepAngle	150

The **Gear** element is intended for emulating gears.



Element's position is assigned by the *Center* and *Radius* properties, setting center coordinates and elements radius. The *Count* property defines amount of the gear claws. The *Depth* property determines claws size. The *DimentionsRatio* defines relative claws width.



The **DigitalText** element is intended for displaying the text.



The text, displayed in the indicator, is assigned in the *Text* property. The position and size are assigned by the *Center* and *Size* properties. The *Angle* property assigns the figure rotation clockwise relatively to the center. The indicator text is displayed by figures, consisted of rectangle segments. The color of figures active segments is assigned by the *ActiveColor* property, the color of inactive segments is assigned by the *InactiveColor* property. The segments vertical and horizontal weight is assigned by the *SegmentSize* property. The space between the segments is assigned by the *SegmentSpaces* property. The text offset relatively to the left element bound is assigned by the *TextHorizontalOffset* property. The text offset relatively to the top bound is assigned by the *TextVerticalOffset*. Besides that, you can set font by the *Font* property.



## Expressions

### Destination and General Principles of the Expressions Use

Expressions can be used for instrument and all its elements. An expression, which calculation result will be bound to the property, can be assigned to the majority of instrument properties. The expressions are used to provide elements interaction and to assign the required instrument functionality.

The order of expressions calculation is the following. If the element state was changed, the expressions recalculation of this element as well as of all elements included in it occurs. The expression calculation result is assigned to the appropriate element properties.

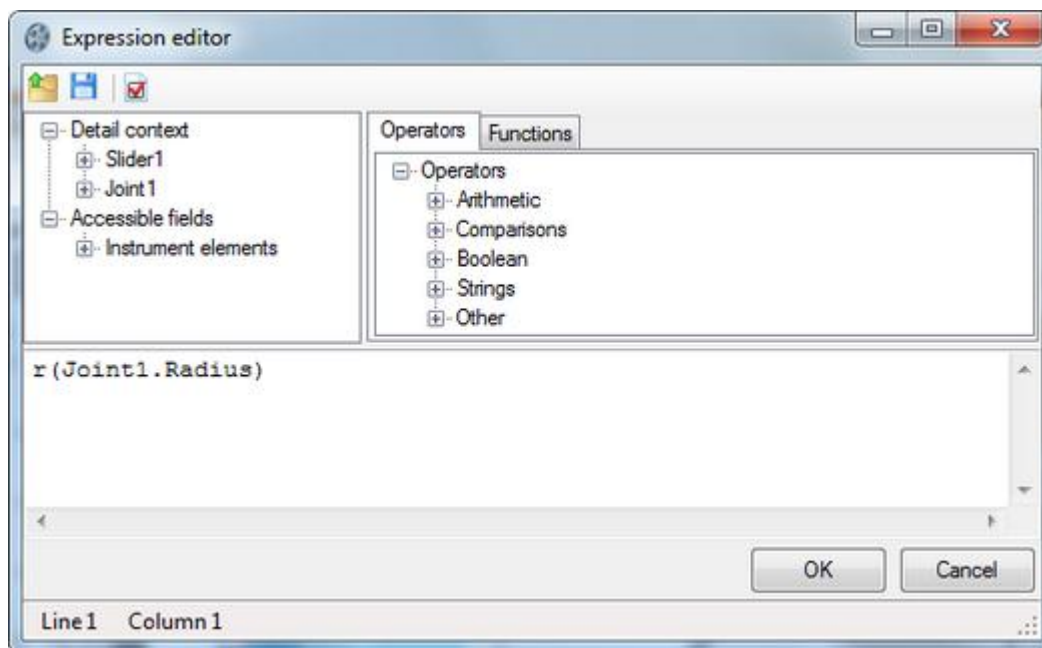
In order to have all instrument element values expressions recalculated during the element state alteration, it is necessary to set the *RecalculateAll* property to "true". The alteration of some properties, assigning the element appearance, doesn't invoke expressions recalculation. For example, the alteration of such properties as Fill, Stroke doesn't invoke expressions recalculation; therefore there is no need to apply to the values of such properties.

If the expression contains syntax, semantic errors, or the exception appears in the calculation process, the expression will be ignored.

Expressions Window for the Circle element:

Active	<input type="checkbox"/>	
BreakEventsBubbling	<input type="checkbox"/>	
Center	<input checked="" type="checkbox"/>	r(Joint 1.Radius) <span>...</span>
Fill	<input type="checkbox"/>	
Radius	<input type="checkbox"/>	
Stroke	<input type="checkbox"/>	
Style	<input checked="" type="checkbox"/>	if(Pressed,"Active"."Inactive")
Visible	<input type="checkbox"/>	

Expression Editor:



### Description of Expression Language Syntax and Semantics

An expression is a task on formula calculation. The expression value is a calculation result. The expression is built out of constants, objects and their properties and methods, operators, function callings and round brackets.





Data types used:

All data types, available in .NET framework, are used in the expressions. Also there is a special support for integer-valued, fractional, logical values, strings and vectors.

The PerpetuumSoft.Framework.Drawing.Vector data type is used for vectors.

The following operations are available in expressions:

### Arithmetic operations

'+' – addition. It is defined for numerical values and vectors.

'-' – subtraction. It is defined for numerical values and vectors.

'-' – unary minus. It is defined for numerical values.

'\*' – multiplication. Numerical values and vectors can be the first argument, the numerical value should be the second argument.

'/' – division. It is defined for numerical values.

'%' – residue of division. It is defined for numerical values.

### Logical operations

'or' – logical OR. It is defined for logical values;

'and' – logical AND. It is defined for logical values;

'=' – equals. It is defined for numerical, logical values and vectors;

'!=' – inequality. It is defined for numerical, logical values and vectors;

'<' – less. It is defined for numerical values;

'<=' – less or equal. It is defined for numerical values.

'>' – more. It is defined for numerical values.

'>=' – more or equal. It is defined for numerical values.

'not' – logical negation. It is defined for logical values.

### Operations over strings

'&' – strings concatenation. It is defined for string values.

The descending priority operations order.

'-(unary), 'not'

'\*', '/', '%'

+', '-', '&'

'>', '>=', '<', '<='

'=', '!='

'and'

'or'



## Constants assigning

<decimal figure>\* - the whole value.

<decimal figure>\*.<decimal figure>\* - value.

For example, 1234 is the whole, 1234.1234 is the fractional value.

For assigning the linear size in generally accepted units the appropriate postfixes at the end of numerical constant are used.

"in" – value in inches

"mm" – value in millimeters

"cm" – value in centimeters

"pt" – value in 1/72 inch points

"px" – value in pixels

For example 3in is 3 inches, 3cm is 3 centimeters.

String constants are assigned with the help of double apostrophes.

For example: "It is a string value".

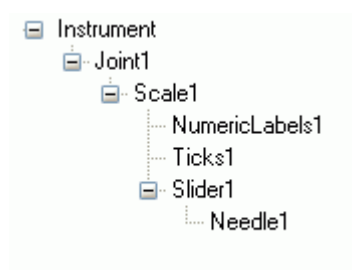
The vector is assigned with the help of "[" and "]".

[<numerical value>,< numerical value>] – vector assigning.

Logical constants are assigned with the help of two key words: 'true' and 'false'.

Access to variables, objects, their fields and methods.

In the expressions, you can apply to variables, available in the instrument by their names. The search of variables and objects by their names is executed in the following way: if an address by the object's name is present in an expression, the GetObject function that should return the desired object is called for the element the given expression is written for. For any element the GetObject function makes the following: the function returns the element itself, in case its name corresponds to the askable one, otherwise it calls GetObject from the parent. Elements-containers execute the search of the desired object through all included elements. Thus, all instrument elements are available in the expressions. For some elements (for example for Slider) the Get Object function returns special variables. According to the rule pointed out above such variables will be available for both element-container and elements included in it. The instrument structure is represented in the picture:



You can apply to the instrument elements (Instrument, Joint1, Scale1, NumericLabels1, Ticks1, Slider1, Needle1) in any element expressions.

Special Slider variables are available in the Slider1 and Needle1 elements expressions.

## Special variables



The following special variables are available in the SliderBase element descendants (e.g., the Slider element):

value – the double type variable. The current Slider value.

hot – the logical type variable. Hot is set to true if the mouse pointer is over an element.

pressed – the logical type variable. Pressed is set to true if the mouse button is pressed when the pointer is placed over an element.

center – the vector type variable. The point that corresponds to the current Slider value and is distant from the trajectory center at a 0 distance.

The following special variables are available in the Joint element descendants (e.g., the Jointelement):

JointRadius - radius of a joint

JointCenter – the vector type variable. The point that corresponds to the current Joint value and is distant from the trajectory center at a 0 distance.

JointTotalAngle - total angle of a joint.

JointStartAngle - the angle of degrees measured clockwise from the X-axis to the start point of the joint.

The following special variables are available in the Guide element descendants (e.g., the Guide element):

GuideLength - length of a Guide.

GuideCenter - center point of a Guide.

GuideEndPoint - end point of a Guide.

GuideStartPoint - start point of a Guide.

For applying to objects' fields and methods the point '.' is used.

For example:

Slider1.Value – applying to the Slider1 element Value property.

Slider1.ToString() – the Slider1 element ToString() method call.

Line1.StartPoint.Rotate(30) – the call of Line1 element StartPoint property Rotate method.

[10, 20].Rotate(30).X – X vector coordination [20, 30], rotated on 30 degrees receiving

ATTENTION! It is impossible to call the object methods, if they are overloaded.

As the expression language is not typified, it is impossible to define what method should be invoked.

In the expressions, you can apply to the functions available in the instrument by their names. The search is executed similarly to the objects search by the names.

The following functions are available for any elements:

The functions of conditional choice:



if (condition, value1, value2), the 1<sup>st</sup> parameter should be of logical type; the 2<sup>nd</sup> and the 3<sup>rd</sup> parameters should be of object type. If the 1<sup>st</sup> parameter is true, the function returns the 2<sup>nd</sup> parameter value, otherwise it returns the 3<sup>rd</sup> parameter value.

switch (condition[0], value[0], ... , value by default), if the condition[i] is set to true, the function returns the value[i], if the condition[i] in any i is false, the value returns by default.

### Formatting Functions

format (object, mask), the object is of object type, the mask is of string type. It returns the object's string presentation according to the mask.

### Mathematical Functions

sin(argument) – calculates the argument sine. The argument is of the double type, the calculation result is of the double type.

cos(argument) – calculates the argument cosine. The argument is of the double type, the calculation result is of the double type.

tan(argument) – calculates the argument tangent. The argument is of the double type, the calculation result is of the double type.

atan(argument) – calculates the argument arctangent. The argument is of the double type, the calculation result is of the double type.

sqr(argument) – calculates the argument to square. The argument is of the double type, the calculation result is of the double type.

sqrt(argument) – calculates the argument's square root. The argument is of the double type, the calculation result is of the double type.

log(argument) – calculates the argument's natural logarithm. The argument is of the double type, the calculation result is of the double type.

exp(argument) – raise E number to the argument degree. The argument is of the double type, the calculation result is of the double type.

sign(argument) – returns: -1 if the argument is negative, 0 – if the argument is 0, 1 – if the argument is positive. The argument is of the double type, the calculation result is of the int type.

abs(argument) – returns the argument's absolute value. The argument is of the double type, the calculation result is of the double type.

round(argument1, argument2) – rounds the 1<sup>st</sup> argument value to the symbols amount after the comma, assigned by the 2<sup>nd</sup> argument. The 1<sup>st</sup> argument is of the double type, the 2<sup>nd</sup> argument is of the int type, the calculation result is of the double type.

The expression can be an argument of any function and method. If the expression calculation result is not appropriate to the argument type, the expression will not be calculated.

### Date Processing Functions

getMonth (argument) – returns the string month name by the date assigned by an argument. An argument should be of the System.DateTime type;

getDayOfWeek (argument) – returns the string day of week name by the date assigned by an argument. An argument should be of the System.DateTime type;



getQuarter (argument) – returns the quarter number of a date specified as an argument. An argument should be of the System.DateTime type;

Other properties and methods of the System.DateTime type can also be used to get other date values.

### The ranking function

Range (currentValue, range, zeroPoint) – extracts data intervals. All arguments should be of numeral type. currentValue – the current value; range – the range; zeroPoint – the zero point.

In the expression language the construction for creation of PerpetuumSoft.Framework.Vector type objects is provided. A vector is created with the help of the following construction:

[<expression>, <expression>]. The creation of objects of other types directly in the expression is impossible. But it is possible to realize custom functions, used in the expressions for the objects of desired type creation.

For the SliderBase type elements (for example Slider) and for the elements included in it, the following special functions are available:

r(radius) – returns the point, corresponding to the current Slider value and distant from the trajectory center (in which the slider is included) at a radius distance.

r(value, radius) – returns the point corresponding to the value and distant from the trajectory center (in which the slider is included) at a radius distance.

a() – returns the slider's rotation angle.

The realization of custom functions and variables, available in the expressions.

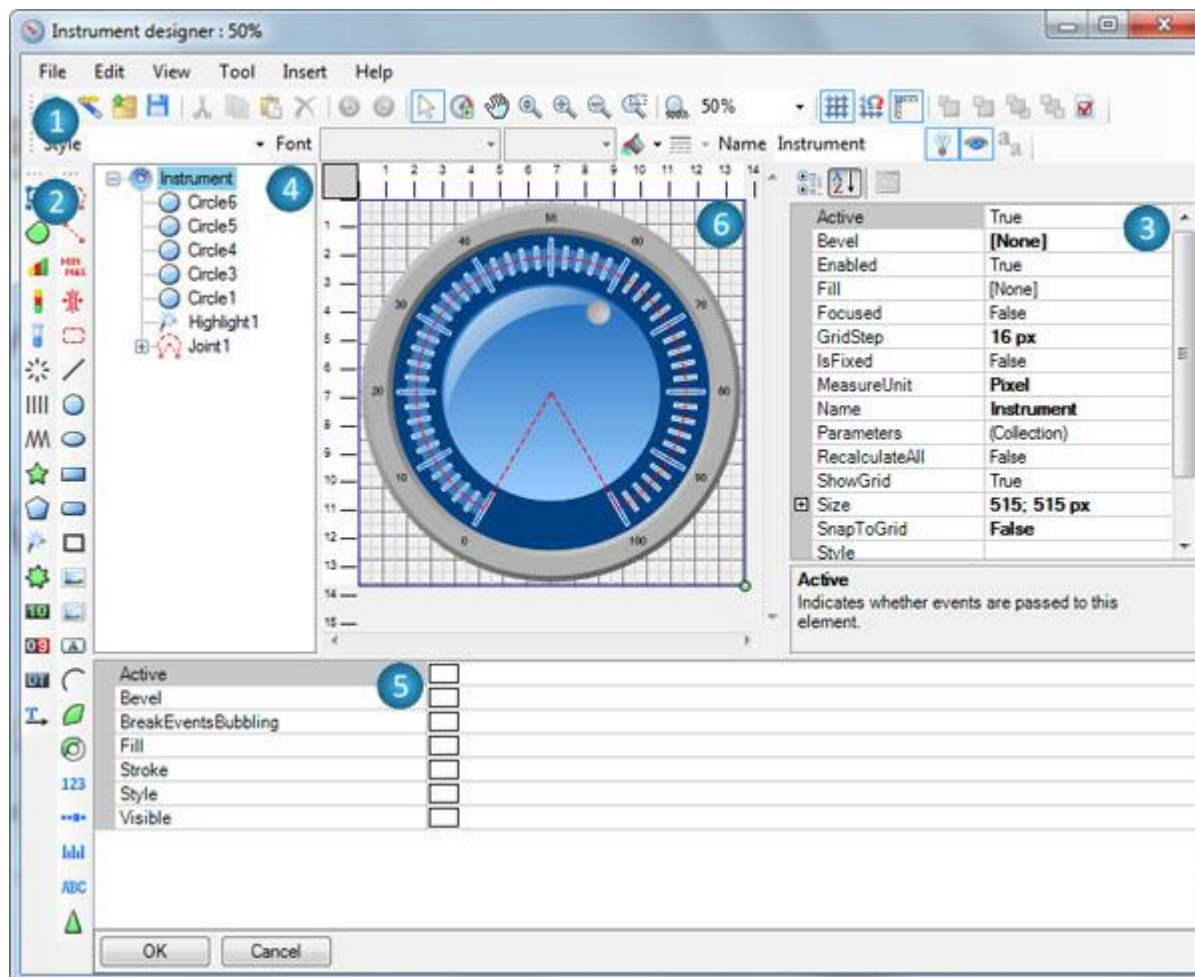
In order to make custom functions and variables available in the element expressions, it is necessary to realize a custom element, which will correspondingly realize the PerpetuumSoft.Framework.Expressions.IExpressionSite interface.

After that you need to register your function. It can be done in the following way:

```
PerpetuumSoft.Framework.Expressions.BuiltInFunctions.RegisterFunction(<Function name>, <implemented IFunction object>);
```

## Use and Capabilities of the Instrument Designer

The SharpShooter Gauges Designer is used for instruments creation. Its appearance is shown on the picture below. An instrument represents an object on which different elements are placed. Each element has a set of properties displayed in the Properties Window. You can edit properties values in this window. Most of properties can be bound to the expressions, assigned in the special expression language. The expressions are displayed in the Expressions Window. You can edit expressions in this window.



- 1 – Toolbar;
- 2 – Elements Toolbar;
- 3 – Properties Window;
- 4 – Instrument Outline;
- 5 – Expressions Window;
- 6 – Work Area.


Visual and non-visual elements for instrument creation in the SharpShooter Gauges Designer are available via the Insert menu or via the buttons on the components toolbox (Elements Toolbox). In order to place an element in the instrument it is necessary to select it from the menu in the Elements Toolbox. Then click the left mouse button in the work area (Work Area), or stretch the element to the desired size by pressing and holding the left mouse button.




The instrument structure is displayed in the Instrument outline window. You can change the elements positional relationship by dragging them with the mouse.

Let's consider the functions available via the File menu. The commands for the work with a file are grouped here.

File\New\Blank menu item (the  toolbar button) – allows creation of a new empty instrument.

File\New\Wizard menu item (the  toolbar button) – allows creation of a new instrument with the help of Wizard.

File\Open Ctrl+O menu item (the  toolbar button) – calls the dialog box for the instrument loading from the file.

File\Open and merge menu item calls the dialog box for the instrument loading from the file. At the same time a new instrument is added to the existing one.

File\Open in wizard and merge menu item calls the Wizard the instrument loading. At the same time a new instrument is added to the existing one.


File\Save Ctrl+S menu item (the  toolbar button) saves the instrument.

File\Save As menu item is used for saving the instrument with a new name.


The File\Export menu item exports an instrument into one of popular bitmapped or vector graphics formats (JPEG, GIF, PNG, BMP, SVG and Macromedia Flash) and calls the save to file dialog box.


You can view the recent files history including the file system paths below the separator.


Let's consider the functions available via Edit menu. The commands for the instrument editing are grouped here.


Edit\Undo Ctrl+Z menu item (the  toolbar button) means the cancel of the changes made in the instrument.

Edit\Redo Ctrl+Y menu item (the  toolbar button) means the repetition of the canceled actions when editing the instrument.

Edit\Cut Ctrl+X menu item (the  toolbar button) means copying and deleting the selected elements.


Edit\Copy Ctrl+C menu item (the  toolbar button) means the copying of the selected elements.


Edit\Paste Ctrl+V menu item (the  toolbar button) means the elements pasting from the buffer to the selected element or its parent, if the selected element is not a container.


Edit\Delete Ctrl+Del menu item (the  toolbar button) is intended for deleting of the selected elements.

Let's consider the functions available via View menu. The commands for the designer appearance control are grouped here.





View\Show rulers menu item (the  toolbar button) is intended for switching on/off the rulers display in the Working area (Work area)


View\Show grid menu item (the  toolbar button) is intended for switching on/off the grid display in the Working area (Work area).

View\Snap to grid menu item (the  toolbar button) is intended for switching on/off the snap of elements' control points to the grid.


Let's consider the functions available via Tool menu. The commands of different tools switching on are grouped here.


The Tool\Select F2 menu item (the  toolbar button) is used to switch on the Select tool. The Select tool allows selecting the elements of the Working area (Work area) and editing the elements via their designers.


Tool\Test F3 menu item (the  toolbar button) allows switching on the Test tool, intended for edited instrument testing.

Tool\Pan F4 menu item (the  toolbar button) is used to panning the picture of the working area (Work area).

Tool\Zoom in menu item (the  toolbar button) is used to zoom an element in.

Tool\Zoom out menu item (the  toolbar button) is used to zoom an element out.

Tool\Region zoom menu item (the  toolbar button) switches on the Region zoom tool to zoom a selected section (region) in.

Tool\Dynamic zoom menu item (the  toolbar button) switches on the Dynamic zoom tool to change zoom of a selected section (region).


Let's consider the functions available via Insert menu. The commands for adding different elements in the instrument are grouped here. Adding commands are broken to groups depending on the element destination.


Elements, grouped in the Insert\Structure menu item, are intended for the instrument structure assigning.

Insert\Structure\Joint menu item (the  button on the Elements toolbox) adds the circular trajectory (Joint element).

Insert\Structure\Guide menu item (the  button on the Elements toolbox) adds the linear trajectory (Guide element).

Insert\Structure\Scale menu item (the  button on the Elements toolbox) adds a scale (Scale element).


Insert\Structure\Slider menu item (the  button on the Elements toolbox) adds a slider (Slider element).

Insert\Structure\Group menu item (the  button on the Elements toolbox) adds a container (Group element), intended for elements grouping.






Insert\Primitives menu item contains the commands of adding the basic primitives, such as geometrical figures and others.

Insert\Primitives\Line menu item (the  button on the Elements toolbox) adds a line (the Line element).

Insert\Primitives\Circle menu item (the  button on the Elements toolbox) adds a circle (the Circle element).


Insert\Primitives\Ellipse menu item (the  button on the Elements toolbox) adds an ellipse (the Ellipse element).


Insert\Primitives\Rectangle menu item (the  button on the Elements toolbox) adds a rectangle (the Rectangle element).


Insert\Primitives\RoundedRectangle menu item (the  button on the Elements toolbox) adds a rectangle with rounded angles. (the RoundedRectangle element)


Insert\Primitives\Frame menu item (the  button on the Elements toolbox) adds a rectangle with a bevel (the Frame element).

Insert\Primitives\Picture menu item (the  button on the Elements toolbox) adds a picture (the Picture element).

Insert\Primitives\PictureSet menu item (the  button on the Elements toolbox) adds a picture set (the PictureSet element).


Insert\Primitives\Label menu item (the  button on the Elements toolbox) adds text inscriptions (the Label element).


Insert\Primitives\Arc menu item (the  button on the Elements toolbox) adds an arc (the Arc element).

Insert\Primitives\Pie menu item (the  button on the Elements toolbox) adds a pie (the Pie element).

Insert\Primitives\RingSector menu item (the  button on the Elements toolbox) adds a ring sector (the RingSector element).

Insert\Scale Elements menu item contains the commands of adding the elements intended for the scales design.

Insert\Scale Elements\ScaleLabels menu item (the  button on the Elements toolbox) – inserts element, intended for numbering scale divisions (the ScaleLabels element).

Insert\Scale Elements\ScaleMarks menu item (the  button on the Elements toolbox) adds graphic markers to the scale (the ScaleMarks element).

Insert\Scale Elements\Ticks menu item (the  button on the Elements toolbox) adds ticks on a scale. (Ticks element).

Insert\Scale Elements\CustomLabels menu item adds text labels to the scale (the CustomLabels element).




Insert\Scale Elements\RangedLevel menu item inserts the RangedLevel element.


Insert\Scale Elements\LinearLevel menu item inserts the LinearLevel element.

Insert\Scale Elements\Tank menu item inserts the Tank element.


Insert\Scale Elements\ScaleTitle menu item inserts scale title (the ScaleTitle element).

Insert\Advanced menu item contains the commands of adding the elements with the specific destination.


Insert\Advanced\Needle menu item (the  button on the Elements toolbox) adds a pointer in the form of a needle (the Needle element).

Insert\Advanced\CircularNotches menu item (the  button on the Elements toolbox) adds the circular notches (the CircularNotches element).

Insert\Advanced\LinearNotches menu item (the  button on the Elements toolbox) adds linear notches (the LinearNotches element).

Insert\Advanced\Spring menu item (the  button on the Elements toolbox) adds a figure in the form of a string (the String element).

Insert\Advanced\Star menu item adds a figure in the form of a star (the Star  element).

Insert\Advanced\Polygon menu item (the  button on the Elements toolbox) adds a regular polygon (the Polygon element).

Insert\Advanced\Highlight menu item inserts the HighLight element.


Insert\Advanced\Gear menu item inserts the Gear element.


Insert\Advanced\Digits menu item (the  button on the Elements toolbox) adds a digital indicator (the Digits element).

Insert\Advanced\Odometer menu item (the  button on the Elements toolbox) adds an element for odometer emulation (the Odometer element).


Help menu item has the About subclause, intended for program data output.

There are also the buttons on the Tool bar intended for the change of output order of the selected elements.

Move selected objects to forward  raises the selected elements up to one position.


Move selected objects to back  lowers the selected elements down to one position.

Send selected objects to forward  raises the selected elements at the very top.

Send selected objects to back  lowers the selected elements at the very bottom.

If the selected elements group order is changed, the relational order inside the group is kept.



The  button is intended for the expressions syntax (used in the instrument) check-up. Check scripts.

Let's consider using the Select tool in the work area.

In the Select tool mode, any visual element can be selected in the work area with a mouse. It is possible to select a group of elements either by means of selecting each element while holding the Ctrl key pressed or by means of "lassoing" the group of elements holding the left mouse button pressed.

Control points in the form of circles are displayed in the selected elements.

The selected elements are highlighted in the instrument structure tree.

The editing operations are applicable to the selected elements:

copy to clipboard (Clipboard);

paste from the clipboard;

deletion;

output order changing;

properties changing with Properties Window use.

the element position changing with the help of the mouse.

changing some element's properties by moving the element control points.

Control points of the selected element are intended for its main properties changing. The point position can be changed by dragging it with the mouse.

Each element has its own control points set. For example, the rectangular elements have ten points, one of which is intended for the center changing, another one is for rotation angle changing, and the rest of them are intended for the corresponding size of the rectangular area changing.

Some elements designers have the control points for the specific properties changing, typical only for the given type elements. For example, the designer for the RingSector.

When the control point position is changed, the appropriate element property also changes. But if the expression is assigned for this property, it makes no sense to change the control point position. Such control points are highlighted with red color. If the designer is able to automatically change the expression that is bound to the property which the control point changes, such point is highlighted with yellow color.

If the element is included in Slider for the current value indication, it is possible to bind the element position to the current Slider value with the help of the designer. To do so it is necessary to click on the control point, keeping pressed the Ctrl key. The expression binding is canceled by the similar action.

The Instrument Outline window displays the instrument current structure in the form of a tree. When placing an element on the Work Area it is automatically displayed in the Instrument Outline. The selected elements are highlighted in the tree with blue color. In this window it is possible to change the elements positional relationship by dragging them with the mouse.

The Expressions window is intended for editing expressions, bound to the appropriate element properties. In the left column, the properties names, for which the expression can be assigned, are displayed. In the right column you can write an expression script or run the script editor. If a script is not assigned for some property, the white blank rectangle is displayed in the right



column. If a script is assigned correctly, the rectangle displays a blue checkmark. If an assigned script contains syntactical errors, the rectangle displays a red criss-cross.

The Properties window is intended for the selected objects properties displaying and editing.

## Work in Windows Forms Applications

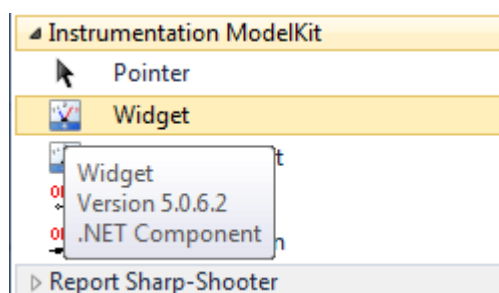
### Components Used in Windows Forms Applications.

For displaying and editing instrument in Windows Forms applications two components are used: Widget and IndicatorWidget. The Widget component is intended for displaying and controlling instrument with the help of the mouse. The Indicator Widget component is a descendant from the Widget. It is intended for displaying instruments, containing the Slider element. Indicator Widget provides additional capabilities for access to the Value property and the ValueChanged event of the indicated slider.

### The Widget Component Use and Capabilities.

Let's consider the use of the Widget component by the example.

Create a Windows Forms application with one form. Then place the Widget component, which can be found on the SharpShooter Gauges tab in the Toolbox, on the form.



The Instrument designer is launched by mouse double click on the component. In the invoked designer, let's open the existing element with the help of the SharpShooter Gauges Wizard. For example, open LightBlueGauge from the Gauge folder. Confirm the changes clicking the "OK" button. The selected element will be displayed in the Widget component.

Run the application for the instrument availability check-up.



The received control element is displayed in the Widget component. It reacts upon the user actions. Now the ability for receiving and setting the current control element value is needed. To do it, let's use the widget1.Instrument property which returns the instrument object. In order to get to the desired element, for example to the *Slider1* or the *Slider2*, let's use the *GetObject* function of the instrument. Then put the Label components onto the form for displaying the control element value in text form.

Add the following lines in the program code. Here the Form1\_Load is the Load event handler of the program main form.

```
private Slider slider1 = null;

private Slider slider2 = null;

private void Form1_Load(object sender, System.EventArgs e)
{
    slider1 = widget1.Instrument.GetObject("Slider1") as Slider;
    slider2 = widget1.Instrument.GetObject("Slider2") as Slider;
    slider1.ValueChanged += new
PerpetuumSoft.Framework.Model.PropertyEventHandler(slider1_ValueChanged);
    slider2.ValueChanged += new
PerpetuumSoft.Framework.Model.PropertyEventHandler(slider2_ValueChanged);
}

private void slider1_ValueChanged(object sender,
PerpetuumSoft.Framework.Model.PropertyEventArgs args)
{
    label1.Text = "Slider1.Value = " + slider1.Value.ToString("0.00");
}

private void slider2_ValueChanged(object sender,
PerpetuumSoft.Framework.Model.PropertyEventArgs args)
{
    label2.Text = "Slider2.Value = " + slider2.Value.ToString("0.00");
}
```

Program work result:

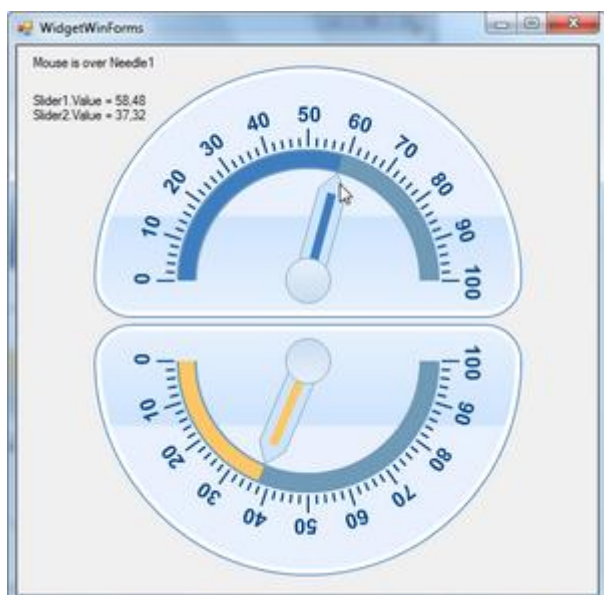


To get the instrument element, which is placed in the assigned point, you can use the *GetElementAt* method.

You can realize the MouseMove event handler in the following way:

```
private void widget1_MouseMove(object sender, MouseEventArgs e)
{
    PerpetuumSoft.Instrumentation.Model.Element element = widget1.GetElementAt(e.X,
e.Y);
    label3.Text = "Mouse is over " + element.Name;
}
```

Program work result:



The Widget component has the following specific properties:

*ZoomScale* – assigns the instrument scale coefficient;

*HideFocusRectangle* – indicates whether the rectangle around the control should be drawn, when it is in a focus.

*InvalidateInterval* – if this value is more than zero, the timer for renewal of the area, occupied by the instrument, will be used.

*InvalidateInterval* assigns the renewal period in milliseconds.

### The IndicatorWidget Component Use and Capabilities.

The IndicatorWidget component use is similar to the one of the Widget component.

But properties, allowing simple receiving of the current value of one of the instrument sliders, are added in the IndicatorWidget.

The *Value* property and the *ValueChanged* event are added in the IndicatorWidget. They correspond to the *Value* property and *ValueChanged* event of one of the instrument sliders used in the IndicatorWidget. In order to assign the exploitable slider, its name is assigned in the *SliderName* property.

Besides, the IndicatorWidget allows moving a specified slider automatically by means of the keyboard.

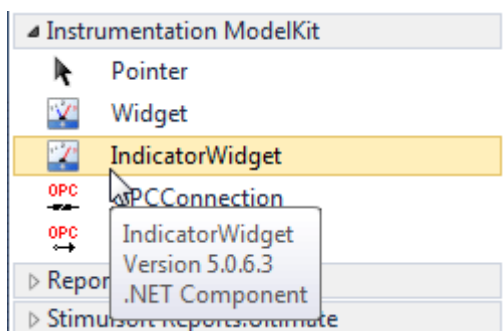
The shift step is assigned in the *Increment* property.

It is possible to change values range for the selected slider scale using the *Minimum* and *Maximum* properties.

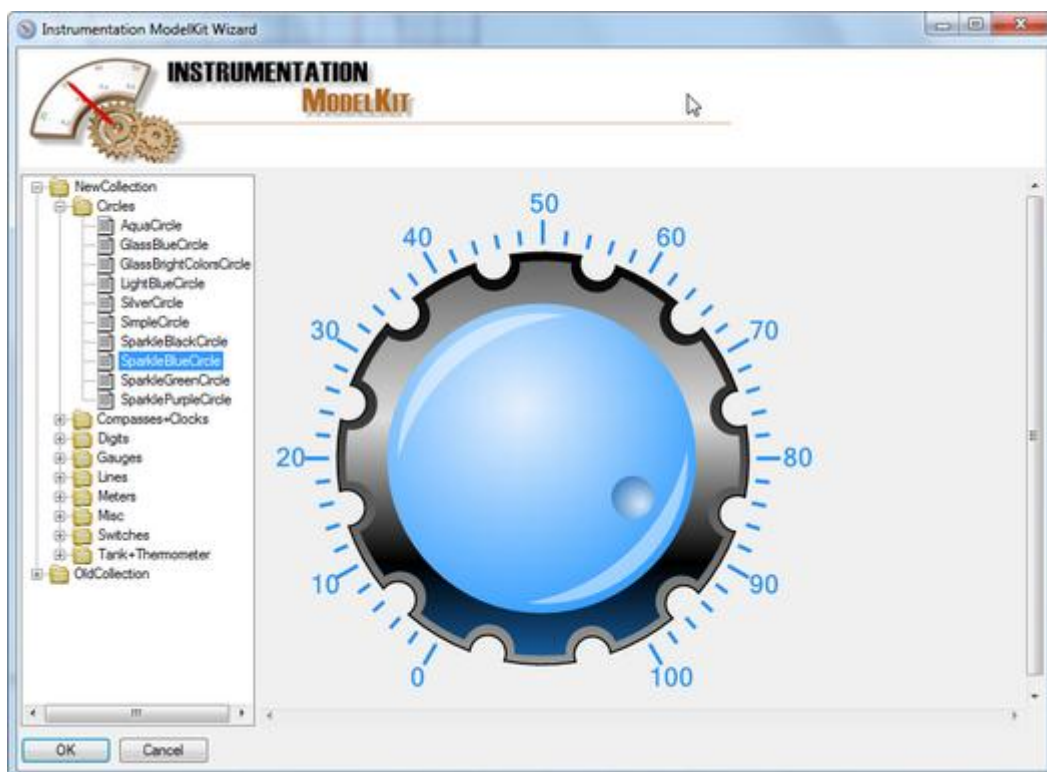
Please, pay special attention to the fact that all above-listed properties have no meaning until the instrument or the desired instrument slider' name for the IndicatorWidget is assigned. If the instrument is assigned, it is more convenient to assign the name of the used slider with the pulldown use in the *SliderName* property. The names of all instrument sliders are displayed in the list.

Let's consider the IndicatorWidget use by the example.

Create WindowsForms application with one form. Put the IndicatorWidget component on the form.



The instrument designer is invoked by the mouse double click on the component. In the invoked designer, let's open the existing instrument with the help of the SharpShooter Gauges Wizard. For example, open SparkleBlueCircle.



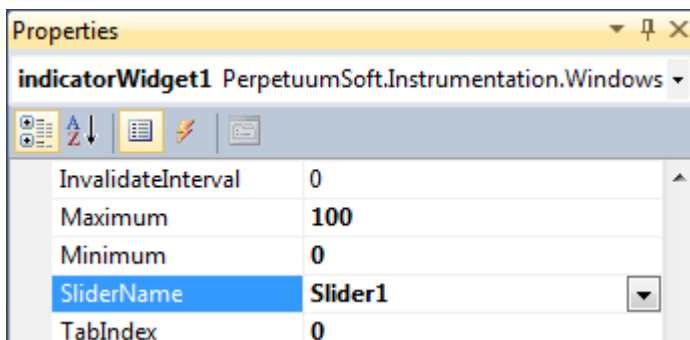
Confirm the changes by clicking the "OK" button. The selected instrument will be displayed in the IndicatorWidget component.

Run the application to check the instrument availability.





Put the Label component on the form for displaying the current control elements value as text. In order to have access to the slider value let's assign the *SliderName* property in Slider1.



Let's realize the ValueChanged event handler in the following way:

```
private void indicatorWidget1_ValueChanged(object sender, EventArgs e)
{
    label1.Text = "Current Value = " + indicatorWidget1.Value.ToString("0.00");
}
```

Program work result:



The access to the current slider value is simplified (compare with the similar example of the `Widget` component use).

So, if you need the access to the current slider value of the instrument slider, it is preferable to use the `IndicatorWidget`.

Now change the scale range, within which the slider is moving, with the help of the *Minimum* and *Maximum* properties. Set the *Maximum* value to 50.



For the convenience of changing slider value from the key board, set the *Increment* property value to 10. Now if you press the "Left" or "Down" buttons, the slider value will be decreased by 10, if you press the "Right" or "Up" buttons, the slider value will be increased by 10.

As the `IndicatorWidget` is inherited from the `Widget`, you can use such properties as: *ZoomScale*, *HideFocusRectangle*, *InvalidateInterval*.



## Work in Web Forms Applications

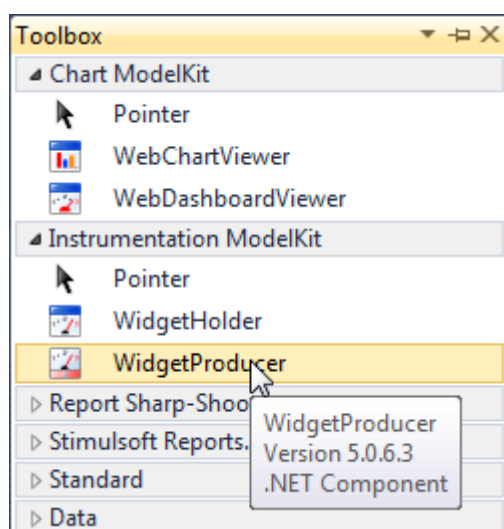
### Components Used in Web Applications.

To display and control an instrument in Web applications two components are used: **WidgetProducer** and **WidgetHolder**. WidgetProducer contains the displayed instrument. This component is placed on a separate page. When this page is applied to, the component substitutes the page content by the instrument image. The WidgetHolder component is used for displaying instrument and for controlling it with a mouse. The WidgetHolder component displays the instrument, contained within the WidgetProducer component.

### The WidgetProducer and WidgetHolder Components Use and Capabilities

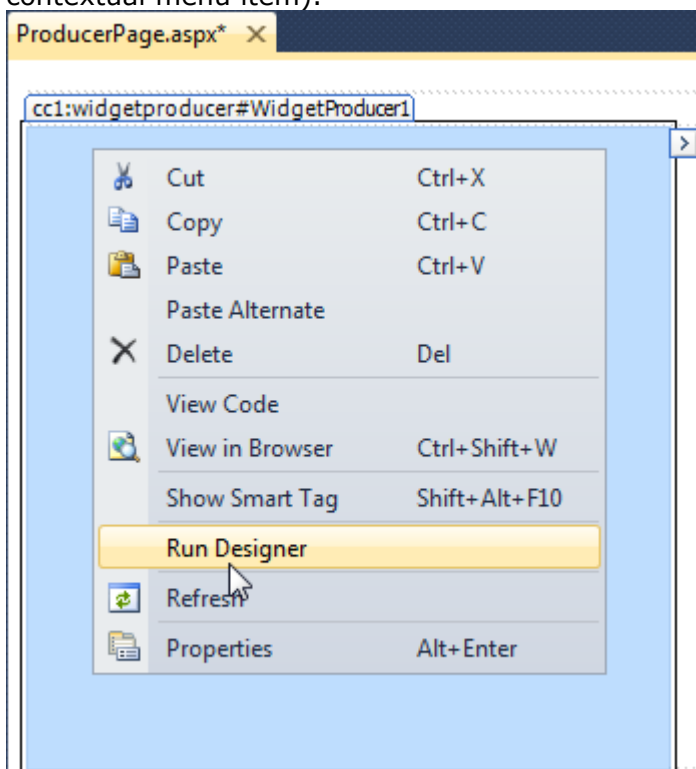
Let's consider the WidgetProducer and WidgetHolder components use by the example.

Create a WebApplication; add the page, set the name ProducerPage. Add the WidgetProducer component.





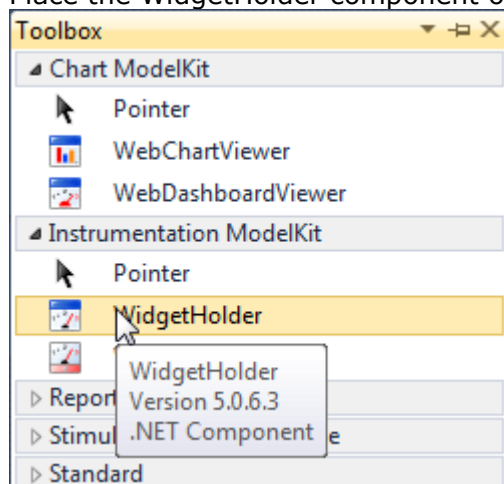
Assign the displayed instrument by calling the instrument editor (“Run Designer” in the contextual menu item).



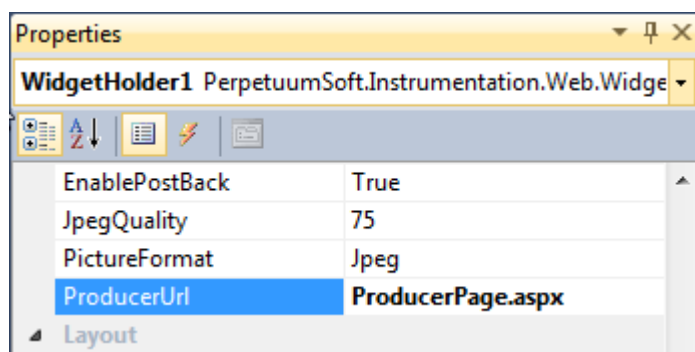
Using the wizard, choose in the designer an instrument which will be used or create a new instrument. By pressing the “OK” button set the instrument for the further use.



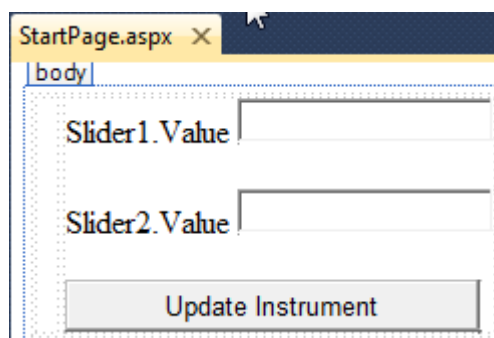
Place the WidgetHolder component on the start application page.



Indicate the address of the page, containing WidgetProducer: ProducerPage.aspx, in the *ProducerUrl* property.



Add control elements on the page for both instrument needles values assigning.



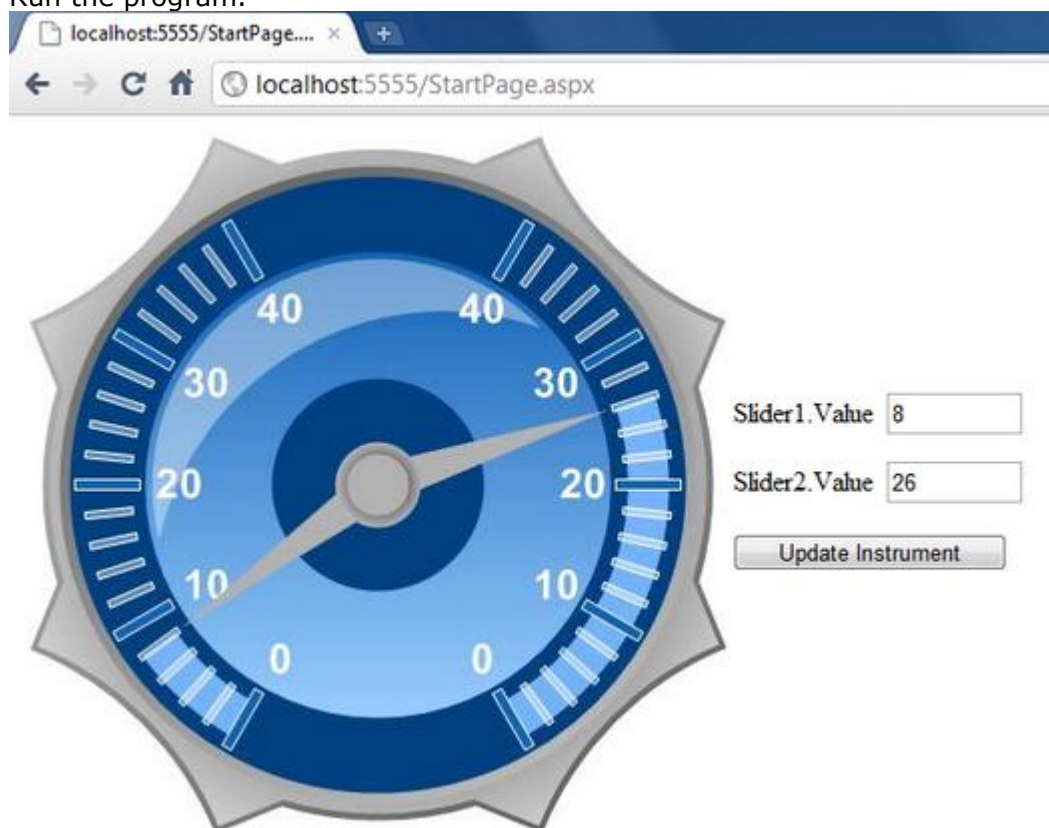
In the upper entry window the left instrument needle value will be assigned. In the lower entry window the right instrument needle value will be assigned. After pressing the button the instrument values will be refreshed.

Realize the Click event handler for the button.

```
protected void Button1_Click(object sender, EventArgs e)
{
    WidgetHolder1.Parameters["Slider1", "Value"] = value1TextBox.Text;
    WidgetHolder1.Parameters["Slider2", "Value"] = value2TextBox.Text;
}
```

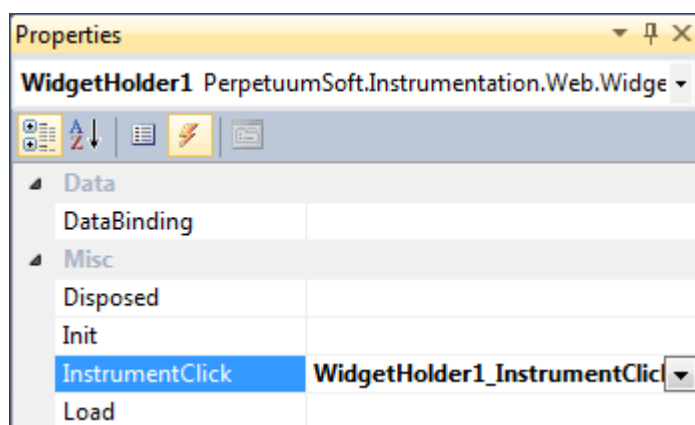
Here the *Parameters* property represents the typed collection. Each collection element represents the "object"- "property"- "value" triplet.

Run the program.



The WidgetHolder component realizes the System.Web.UI.IPostBackEventHandler interface. If the *EnablePostBack* property is set to "True", the mouse click on the instrument will be handled in the InstrumentClick event.

Let's consider it by the following example. Realize the InstrumentClick event handler for the WidgetHolder component.



```

protected void WidgetHolder1_InstrumentClick(object sender,
PerpetuumSoft.Instrumentation.Web.InstrumentClickEventArgs args)
{
    PerpetuumSoft.Instrumentation.Model.Instrument instrument =
args.Element.Instrument;
    PerpetuumSoft.Instrumentation.Model.Scale scale = instrument.GetByName("Scale1")
as PerpetuumSoft.Instrumentation.Model.Scale;
    WidgetHolder1.Parameters["Slider1", "Value"] = scale.PointToValue(new
PerpetuumSoft.Framework.Drawing.Vector(args.X,
args.Y)).ToString(System.Globalization.CultureInfo.InvariantCulture);
}

```

## The Technology of the Instrument Design.

### Designing the Simple Instrument with a Slider

Let's consider the instrument designer utilization for designing instruments.

Creating the instrument with a slider.

Let's determine our requirements to the designed instrument.

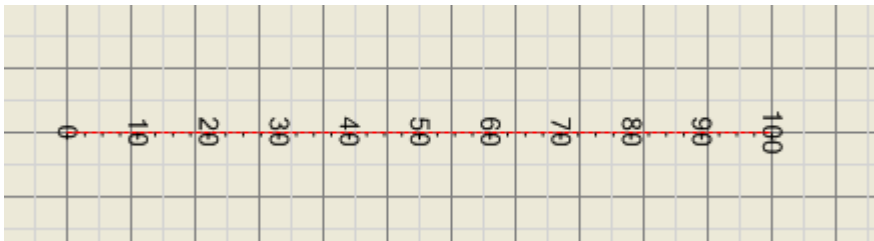
1. The instrument should contain a slider moving on the scale within the limited range from 0 to 100.
2. Ticks on the scale should be designed in different colors within the following ranges: [0-80) and [80-100).
3. When the slider exceeds the 80 mark it should change the color.
4. When the mouse is on the slider, the current value should be displayed alongside as text.

1. The instrument should contain a slider moving on the scale within the limited diapason from 0 to 100.

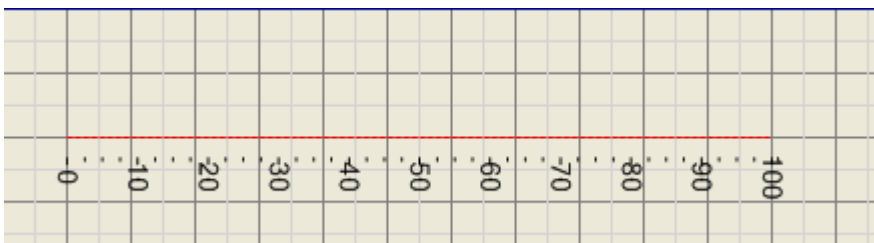
Place the linear trajectory into the instrument. The trajectory is intended for assigning the scale shape. Place the Scale element into the instrument. The scale assigns value range. To make scale lie on the trajectory you should place it into the trajectory (you can do it by dragging it in the Instrument elements tree).


To visualize the scale element you should mark it by ticks and text marks. So add the Ticks and ScaleLabels elements from the Element toolbox into the Scale1 element.

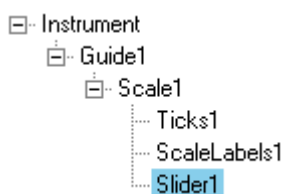
The instrument itself.




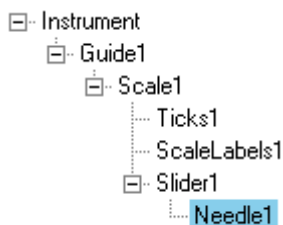
Let's set the Distance property for the Ticks1 to 10. Let's set the Distance property for the NumericLabels to 20.



Add the Slider element (  ), intended for representing the current value and its changing with the help of the mouse, into the instrument. The instrument structure is as follows.



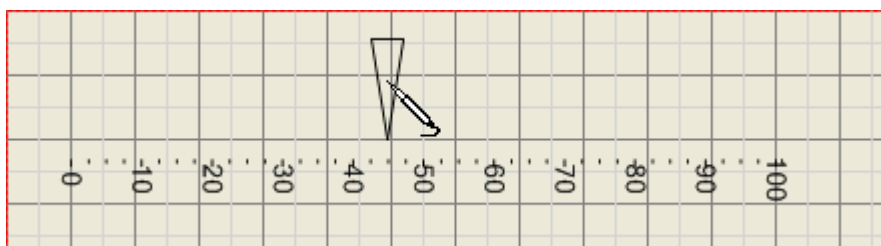
To visualize the slider you should add the Needle element (  ) into the Slider1.



Let's bind Needle1 to the slider value. To do it you should write the following in the Needle1 expressions:

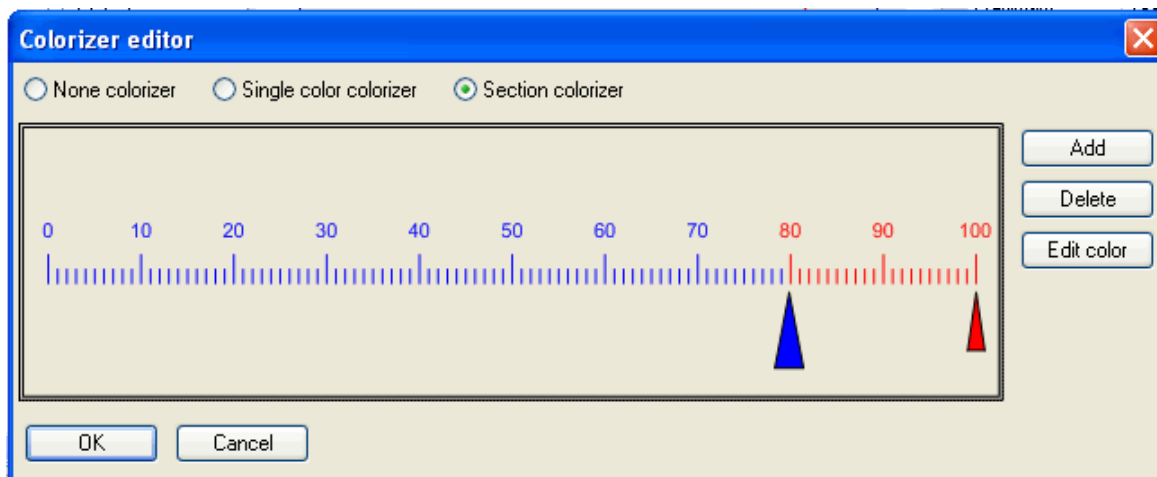
"r(0px)" expression for the StartPoint and "r(-50px)" expression for the EndPoint.

Now the slider is able to move with the help of the mouse.



2. Ticks on the scale should be designed in different colors within the following ranges: [0-80) and [80-100).

Change the Colorizer property for the Scale1 element using the Colorizer Editor.



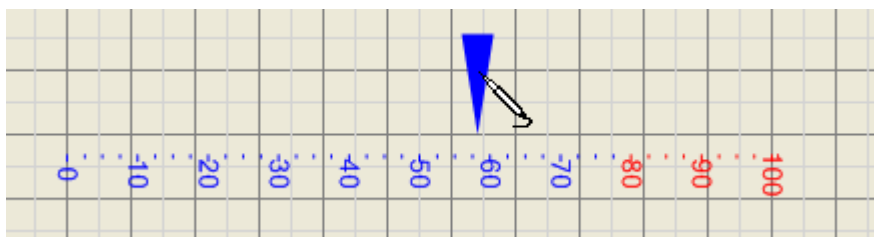
After all changes the instrument should have the following look.



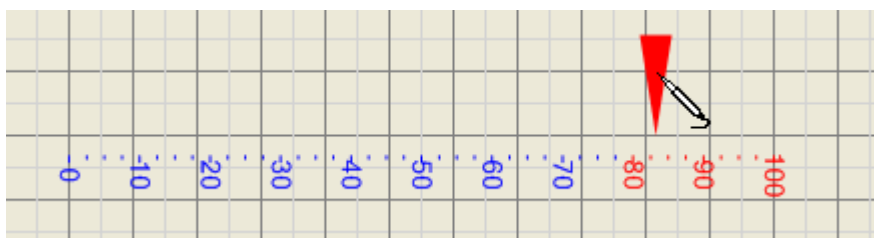


3. When the slider exceeds the 80 mark it should change the color.

Add two styles in the instrument styles collection (InstrumentStyles): ColdColor style with blue fill and HotColor style with red fill. Write the following expression for the Style property of the Needle1 element: `if (Slider1.Value < 80, "ColdColor", "HotColor")`.



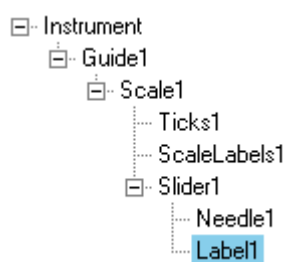
the current value is `<` 80



the current value is `>` 80

4. When the mouse is on the slider, the current value should be displayed alongside as text.

Add the Label element ( **A** ) into the Sleder1 element.



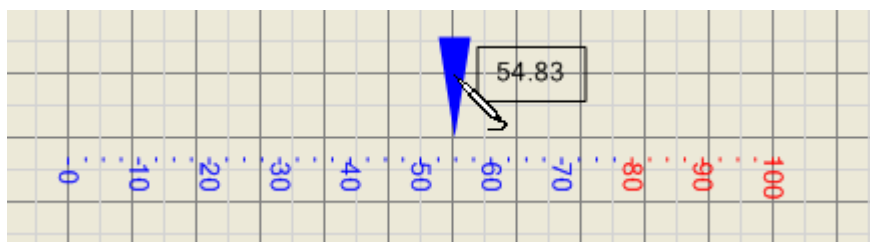
Let's set the BreakEvenBubbling property of the Label1 element to True. This assigns mouse events not to be passed from Label1 to Slider1.

Let's write the following expressions for the Label1 element:

Write `"r(-38px) + [120, 20]"` expression for the Center. Label1 center will be set depending on the slider value and will be shifted on some distance.

Write `"format(value, "0.00")"` for the Text. The Label1 element will display the slider current value formatted according to the mask.

Set Visible to "hot". The Label1 element will be viewed only when mouse is on the Needle1.



The sample with this instrument can be viewed in the HorizontalSlider.imk file contained in the instrument designer.


### Creation of Instruments Using Complex Expressions


Let's create an instrument, in which the slider will move within a plane.


Firstly, let us define requirements to the instrument.

1. The instrument should comprise two datum lines.
2. The slider should move within a plane.
3. It is necessary to display the slider center projection on each of the datum lines.
4. The text representation of the slider position on the plane should be displayed beside the slider.

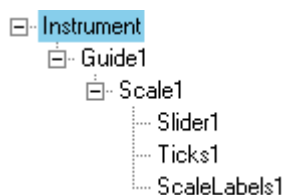
1. The instrument should comprise two datum lines.

Let's place the Guide trajectory (  ) into the instrument. The trajectory is intended for assigning the scale form.

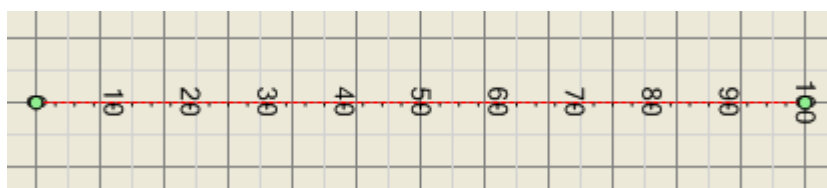
Let's place the Scale element (  ) into the instrument. The Scale assigned the values range. In order to make the Scale lie along the trajectory, it is necessary to place it into the trajectory (simply drag-and-drop it in the object tree).

Let's add the Slider element (  ), which is intended for representing the current value and changing it by means of mouse move.

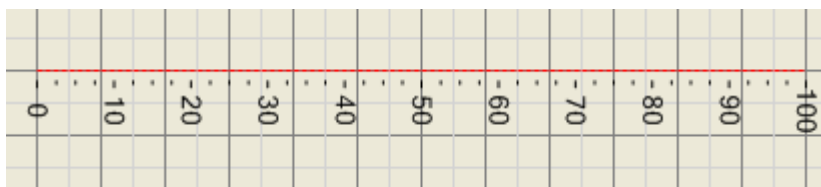
To visualize scale, let us place the ticks and text markings on it. To do that, we should add the Ticks and NumericLabels elements from the Element toolbox into the Scale1 element. After that, the element structure should appear as follows.



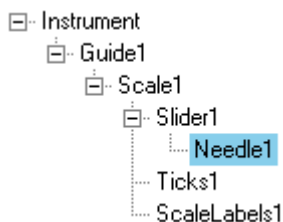
The Instrument itself.



Set the Ticks1 Distance property to 10. Set the NumericLabels1 Distance property to 20.



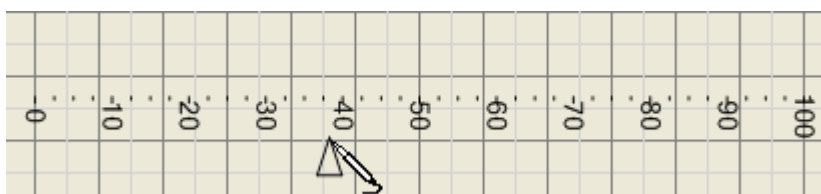
To visualize the slider, add the Needle element ( ) to the Slider1 element.



Let's set the Needle1 element Stroke property to SimpleStroke.

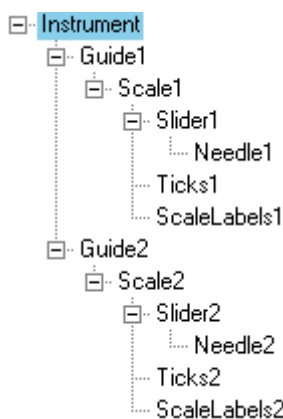
And now let us bind the Needle1 element to the slider value. In the expression window for Needle1 write the following: "r(1.3cm)" for StartPoint and "r(0.8cm)" for EndPoint.

Now the slider should be capable of moving when you manipulate it with a mouse.

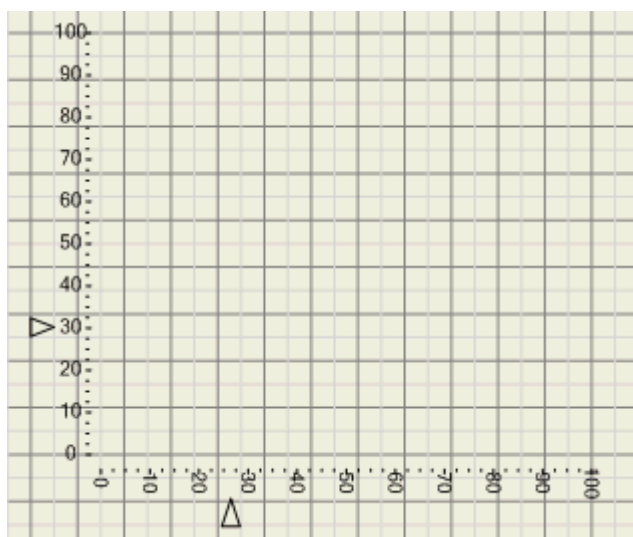


As a result we shall get the OX axis.


Let us repeat the above mentioned actions for assigning the other linear trajectory, its scale and slider. We shall get the OY axis. Then the element structure should appear as it is shown below.

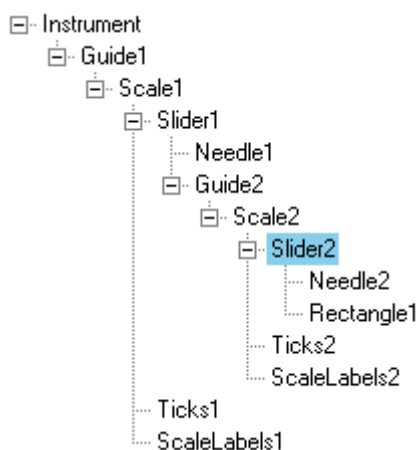


The Instrument itself.



2. The slider should move within a plane.

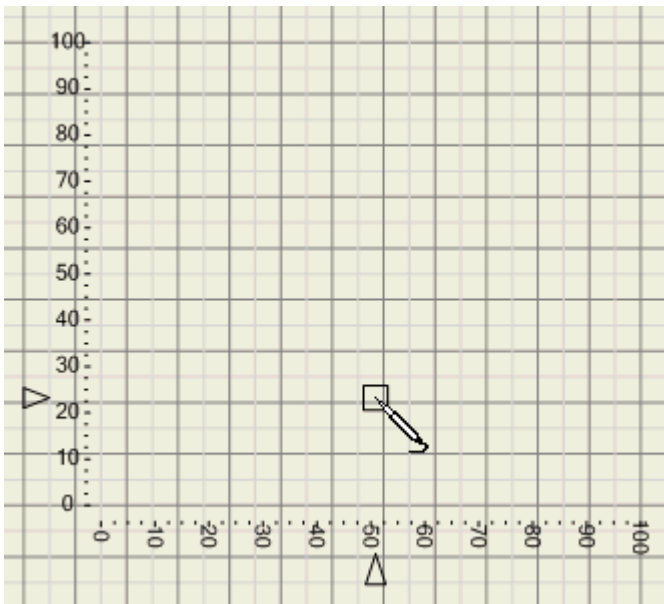
Let us create a slider common for OX and OY axes. To do that, we shall add the Rectangle element (  ) to Slider2, having set the Stroke property of the Rectangle element to SimpleStroke. At the same time we shall place the Guide2 element into Slider1. After we do that, the element structure will be changed as follows.




Now let us write the “[Slider1.GetPosition(0).X, Slider2.GetPosition(0).Y]” expression for the Rectangle element Center property.

The Slider GetPosition function returns the point which corresponds to the value assigned by the argument.

Now the values of Slider1 and Slider2 will vary when the Rectangle1 element is moved by the mouse.




3. It is necessary to display the slider center projection on each of the datum lines.

Let us add the Line1 element (  ) into Slider1. We shall define the Line1 Stroke element as SimpleStroke and choose red line color in the Stroke properties. For the Line1 element we shall use the following expressions:

"r(0.42cm)" for StartPoint; and

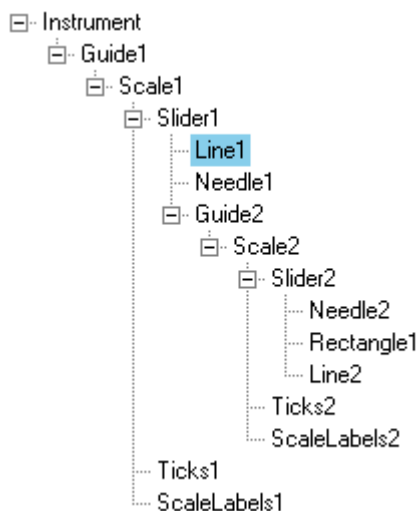
"r(-5.93cm)" for EndPoint. We shall get the OX axis projection.

Let us add the Line2 element (  ) into Slider2. We shall define the Line1 Stroke element as SimpleStroke and choose blue line color in the Stroke properties. For the Line2 element we shall use the following expressions:

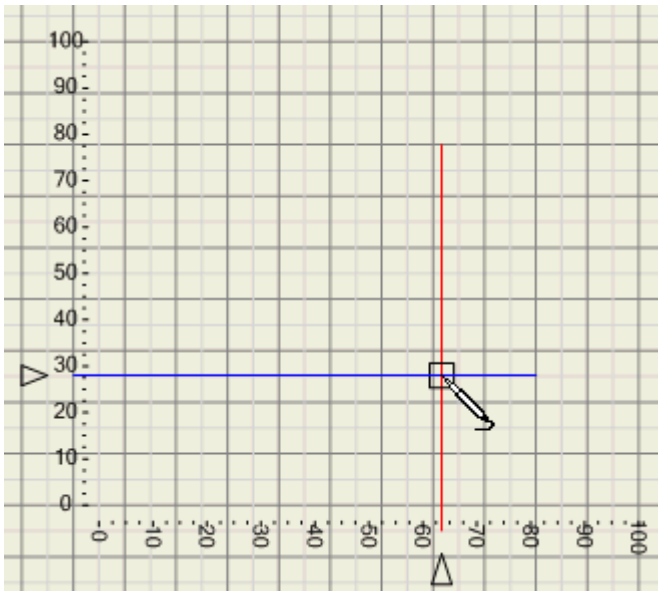
"r(7.2cm)" for StartPoint; and

"r(-0.42cm)" for EndPoint. We shall get the OY axis projection.

As the result the element structure will take the following look.

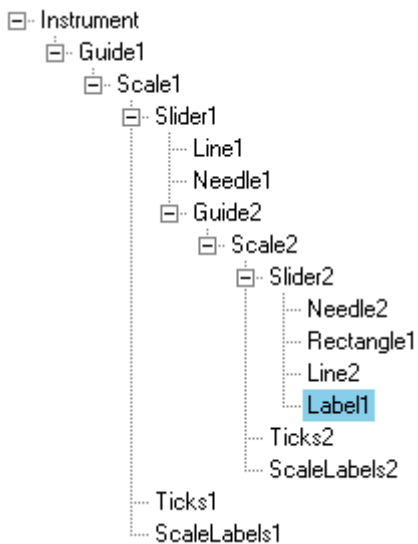


The instrument will look as follows:



4. The text representation of the slider position on the plane should be displayed beside the slider.

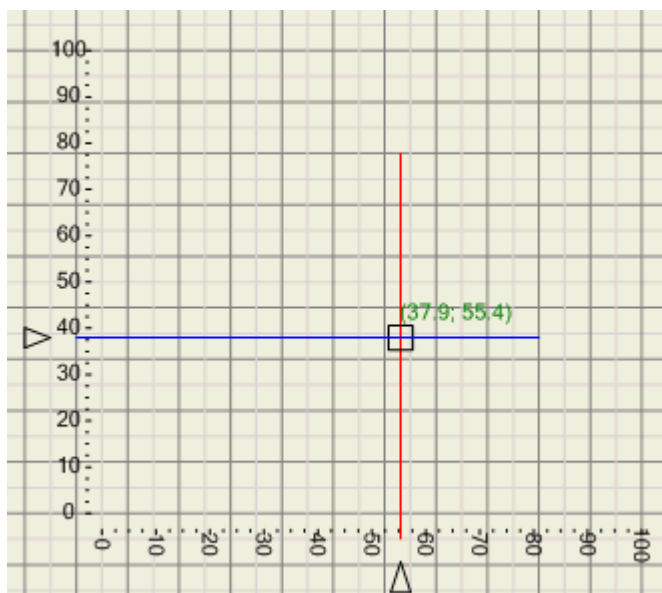
Let's create the Label element ( **A** ) within the Slider2 element.



Let us indicate green color as a fill color in the Label1 element Fill property. For the Label1 element we shall use the following expressions:

"[Slider1.GetPosition(0).X, Slider2.GetPosition(0).Y] + [35px, -15px]" for Center; and

" (" & format(Slider2.Value, "00.0") & "; " & format(Slider1.Value, "00.0") & ")" " for Text.



You can find the instrument we finally got if you open the Gauge2D.imk file in the instrument designer.

## Working With the OPCChannel Component

### General Information

The SharpShooter Gauges provides the capability to exchange data with OPC-servers. OPC is the industry standard for interacting with real time devices. Utilizing the PerpetuumSoft OPCConnection and OPCChannel components, one can perform data exchange without resort to use complicated COM interfaces.

### OPCConnection

The OPCConnection component is intended for producer-supplied device COM-server connection setup. The design time adjusting is conducted by means of a convenient standard interface. It is enough to place the component onto the form and call the connection setup window in a corresponding PropertyGrid.

The window displays local OPC-servers and enables to connect to any server available within a local network (requires necessary permissions including DCOM permissions – please consult your system administrator).

### OPCChannel

The OPCChannel component is intended for adjusting the selected connection data transmission channel. Please note, that it is necessary to set the required OPCConnection for the corresponding component by means of the OPCConnection property in the PropertyGrid before adjusting data transmission channel. The connection to an OPC-server takes place when the OPCConnection is being set up at design time or by user demand at run time (the EstablishConnection method).

After connection setup, a user can select any number of positions available on a server with the help of the Items property in the PropertyGrid.

After a position is selected, this position properties such as ID, access rights and canonical data type are displayed.

The canonical data type is the standard CLS-compatible .NET type:

Reading/writing from/into a data channel is handled by the corresponding OPCChannel public methods:

AsynchronousRead – asynchronous data slice read;



AsynchronousSubscription – asynchronous subscription to a data slice (s) with the assigned frequency and the number of slices in one event;

Write – write.

The results of any operation are returned asynchronously; in this case OPCChannel generates an event. At the transmission of data received by asynchronous read or subscription (AsynchronousRead and AsynchronousSubscription correspondingly), a DataTransmission event the arguments of which contain the requested data together with the ITransaction transaction descriptor is generated. The WriteStatus event is used to return the Write operation results.

In order to handle events it is necessary to specify a corresponding event handler like

```
opcChannell.DataTransmission += new  
PerpetuumSoft.Framework.Connectivity.DataTransmissionEventHandler(this.handler);
```

Please look up in the ClassReference and utilization examples included in the delivery package for more information. For proper sample operation installation of Matrikon OPC Simulation Server is required.

### Step-by-step Example of Application Creation

Let us consider the use of the OPCChannel and OPCConnection components by an example. Launch a WindowsForms application with one form. Then place the IndicatorWidget component which will display the derivable values and the OPCConnection situated in the SharpShooter Gauges onto the form.

Select an OPC-server to connect to with the help of the Server property in the PropertyGrid. In the dialog window select one of OPC-servers installed on your local host or local network.

Now it is necessary to place the OPCChannel component situated in the SharpShooter Gauges tab onto the form.

Let us bind the OPCChannel to a preliminarily specified connection. To do that, set the OPCConnection property in the PropertyGrid.

Now we need to assign the OPC-server positions to receive the data. It is necessary to set the OPCChannel Items property in the PropertyGrid. You can select any number of positions, but it is important not to allow the replication of positions in each OPCChannel.

Now your channel is tuned, but you need to dispatch a corresponding query to the channel in order to read data. It is enough to call the AsynchronousSubscription method at form upload or by user demand. The primary variant of this method takes one parameter – the event interval in milliseconds.

```
private void Form1_Load(object sender, System.EventArgs e)  
{  
    opcChannell.AsynchronousSubscription(500);  
}
```

The next step is to create an event handler for the DataTransmission event which occurs when the OPCChannel receives data from the OPC-server. To do that, you should double click the Events tab in the PropertyGrid and add a new method (the DataTransmission event handler).

```
private void opcChannell_DataTransmission(object sender,  
PerpetuumSoft.Framework.Connectivity.DataSlicesEventArgs e)  
{  
    DataSlice slice = e[0];  
    double d =  
System.Convert.ToDouble(slice.GetDataPoint(opcChannell.Items[0].Item.ItemID).Value  
);
```





```
    this.indicatorWidget.Value = Math.Abs(d);  
}
```

In this case, the first position value of the first data slice will be accepted at each event. The multiple data slices handling is intended for work with the extended version of the `AsynchronousSubscription` method that takes the data update frequency as well as data transmission frequencies.

When you launch your application, you'll see the OPC-server data displayed by an instrument in the real-time mode.

## Advanced DataChannel Design

### Capabilities

The DataChannel concept underlying the DataChannel component allows a developer to create real-time data transmission channels using virtually any data provider including the analytically specified functionals.

### Design

To create a custom DataChannel implementation, a developer should only create a component class inherited from an abstract DataChannel and a component class that will inherit an abstract Connection. There is no need to functionally implement all methods and properties in order to implement a functional DataChannel. Many methods and properties providing additional services (`CultureInfo`, `ProviderCurrentTime` etc.) might be non-implemented.

The OPCChannel is an example of a DataChannel implementation. It is useful to explore its source code before implementing a custom component.



## SharpShooter OLAP

### Destination and Basic Features

SharpShooter OLAP is intended for developing and processing cross reports based on high-volume statistical data. Nowadays, managing data comprised by stacks of statistical information seems to be an unsolvable task without any automation. At the same time, the demands for dynamic change of data presentation based on one and the same data set often lead into a dead end. SharpShooter OLAP is a convenient solution to deal with such a challenge.

A full-featured graphical designer destined for pivot table retrofitting allows visualized indication of the data which is to be used for report creation. In order to get a ready-made report it is often enough to simply drag the required fields given within a certain dataset into the corresponding fields in the designer and hit a couple of keys to set a representation style.

Viewing the same data taking into account all possible dimensional groups is simple as well. The only thing that has to be done is dragging a group header into a required table field.

In order to assign value calculation regulations for both table dimensions and cross dimensional cells one can use various expressions. It allows adjusting a table in accordance with the information contained in a data set. The use of expressions affords a wide range of opportunities from basic summation of several data set field values up to various sorts of groups within a desired report (like fragmentation of date into years, months, quarters and days of week or indication of value intervals within a group).

To calculate pivot table cross-dimensional values there can be used one of aggregate functions (e.g. mean value calculation or the search for maximal/minimal value).

You are given an opportunity to choose one of preset pivot table presentation styles or utilize XP style. You can also create your own styles which can be saved to a file for further use.

The use of a generated pivot table is not limited by the borders of your application. It is possible to print a cross report or export it into an Excel table, PDF or HTML document where all current pivot table settings (such as dimensional groups minimized by a user, data sorting and filtering, or presentation style) will be taken into account

All pivot table settings can be saved to a file and this makes it possible to quickly go back to the unconfigured state of a table at any moment.

### Getting Started

The DataCube component is intended for organizing the data from an outer data source as a pivot table. To specify a table, fields for each dimension (by rows and columns) should be specified. One should also specify the fields (facts) from which the data are taken to calculate values on intersections of rows and columns.

A regulation of data access is assigned in a field with the help of an expression.

Simple arithmetical and logical operations are available in expressions. Source fields can also be used as variables in expressions.

For the fact fields there can also be assigned an aggregation function used to aggregate data on rows and columns crossovers.

The DataCubeGrid component is intended for displaying the data presented in the DataCube. To do this, it is necessary to specify the Source property for the DataCubeGrid.

The current section contains a simple step-by-step example of SharpShooter OLAP utilization. Let us create a pivot table representing the data related to sales total of a certain product for each of several companies.



The initial data table structure is shown below.

Field name	Type	Description
CategoryName	String	Name of food category.
ProductName	String	Name of product
CompanyName	String	Name of company
UnitPrice	Currency	Unit price
Quantity	Number	Quantity
Discount	Number	Discount

The grouping of data in the resulting table is presented in the image below.

		CompanyName	
CategoryName	ProductName	Company1	Company2
Category1	Product1	<i>SalesSum11</i>	<i>SalesSum12</i>
	Product2	<i>SalesSum21</i>	<i>SalesSum22</i>
Category2	Product3	<i>SalesSum31</i>	<i>SalesSum32</i>
	Product4	<i>SalesSum41</i>	<i>SalesSum42</i>

**SalesSum = UnitPrice \* Quantity \* (1 - Discount)**

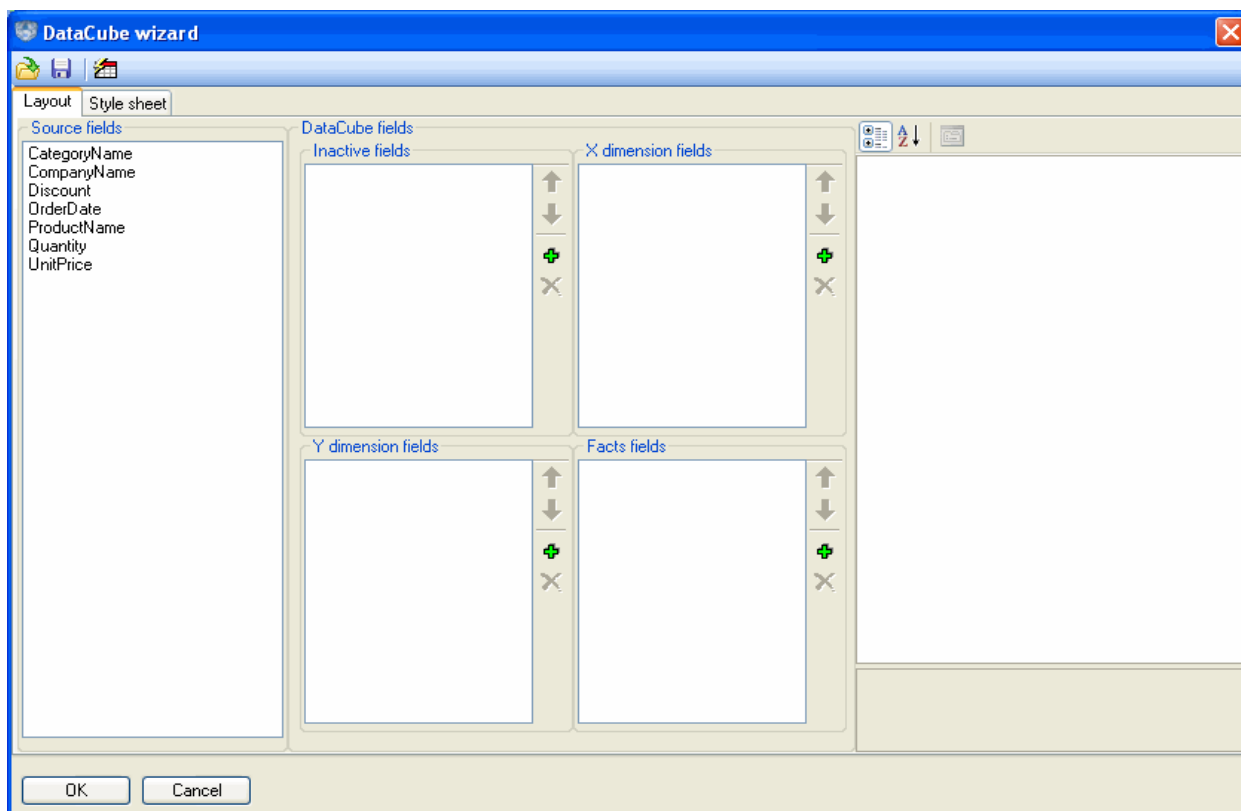
The columns will contain the names of supplier companies and the rows will contain product categories and denominations. The intersections will provide total sales information subject to products prices, quantity and discounts.

We shall use the demonstrational database Access NWind.mdb as source data.

Run Microsoft Visual Studio and create a new C# Windows Application project.

For a start let's create a data source for our report. To do that, add the System.Data.DataSet object onto the form and create a Sales table with fields corresponding to the initial data table structure.

Add the DataCube object onto the form. Indicate the Sales table from the DataSet as the DataSource value. Run the DataCube object designer by means of clicking Run Designer in its property window.



Drag the CompanyName caption from the Source fields list into the X dimension fields list. Add CategoryName and ProductName from the Source fields list into the Y dimension fields list.

Click the "Add" button in the Facts fields list. Indicate the following properties for a new field:

Expression  $\text{UnitPrice} * \text{Quantity} * (1 - \text{Discount})$

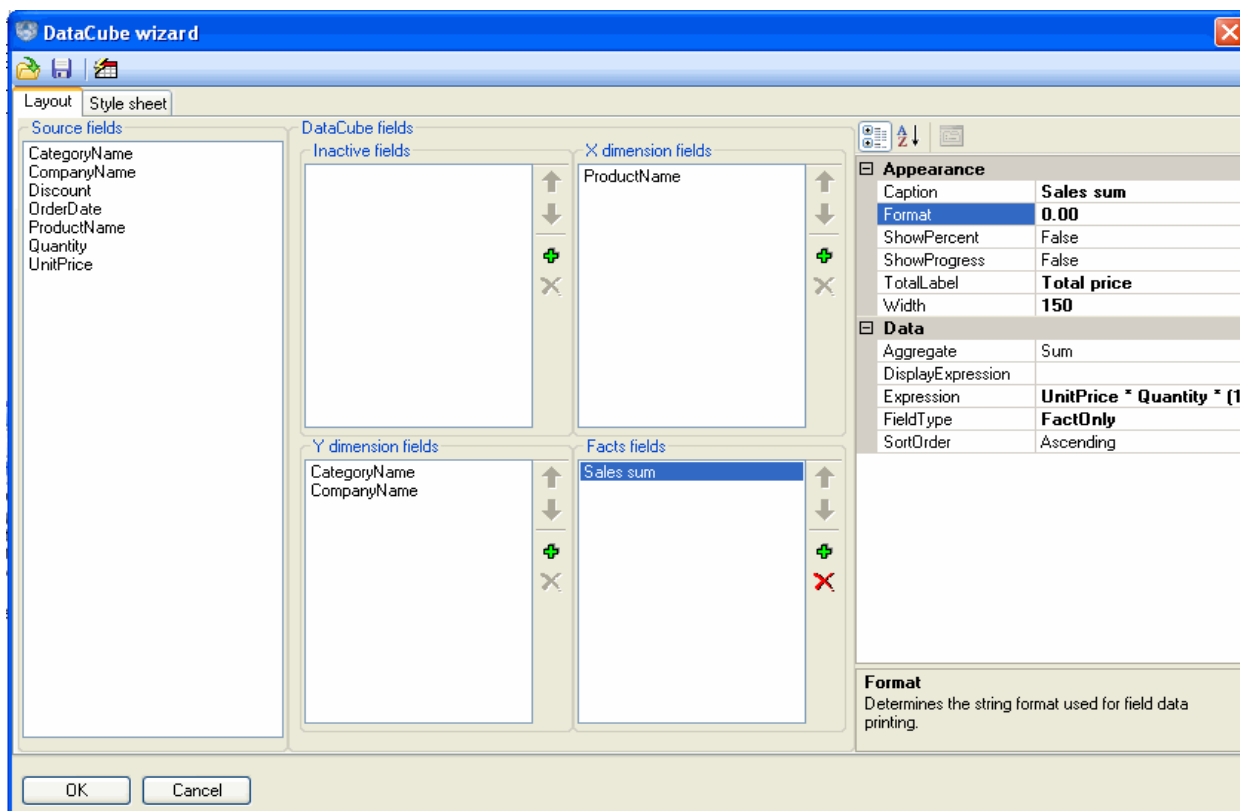
Caption Sales sum

Field type FactOnly

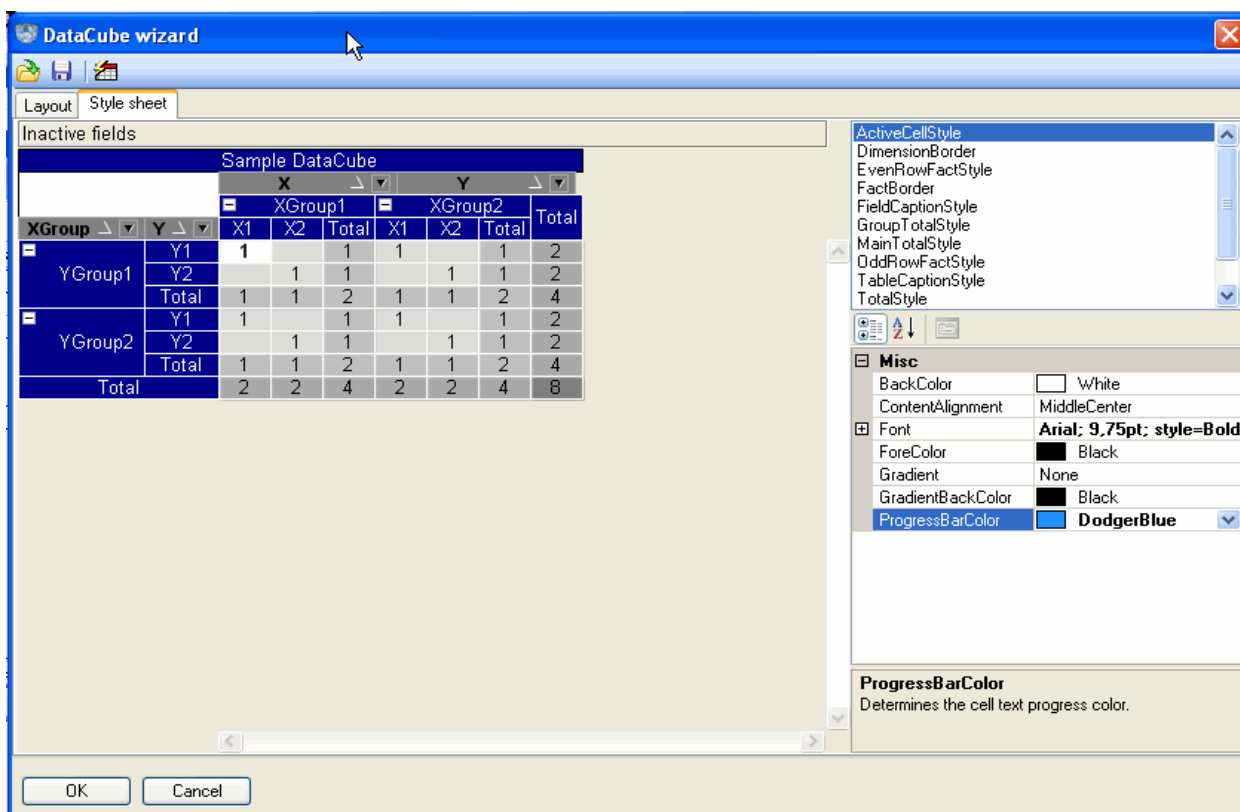
Format 0.00

Width 150

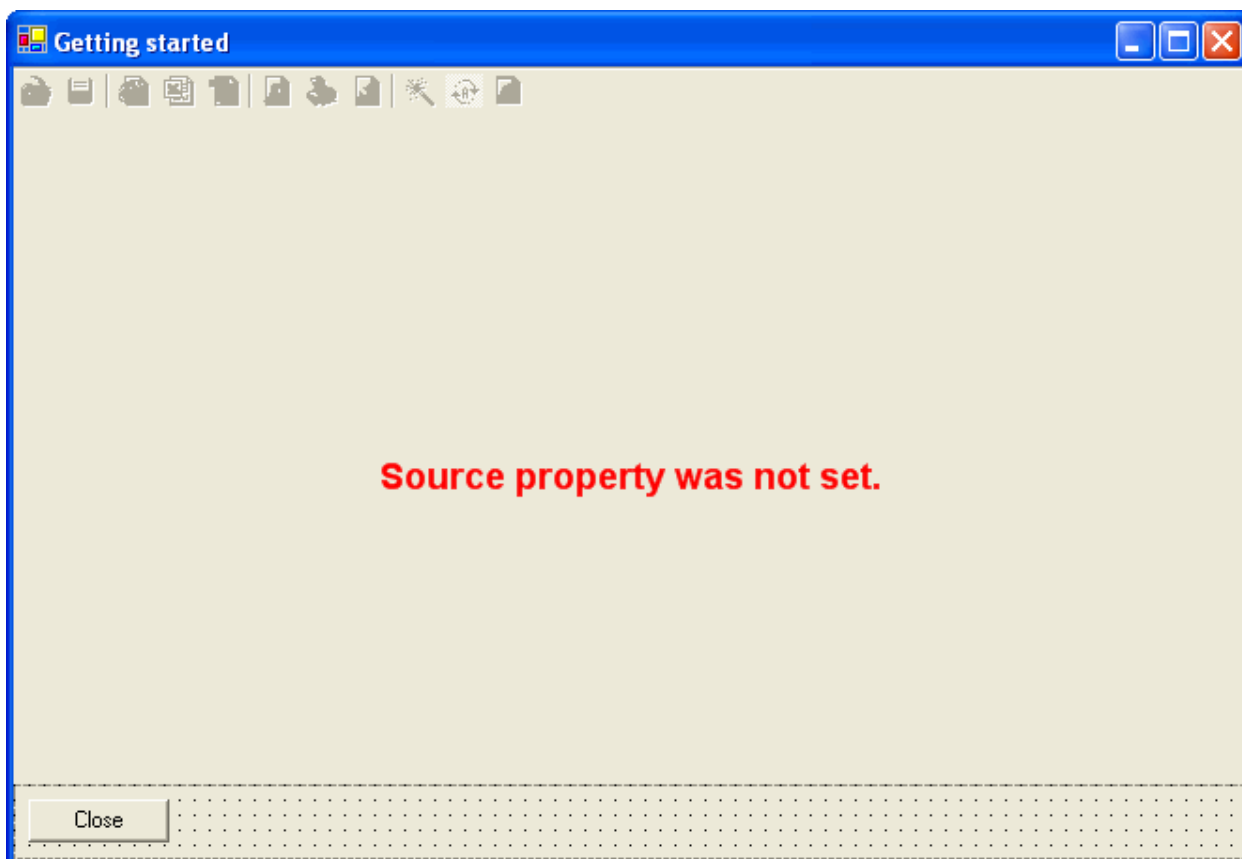
Total label Total price



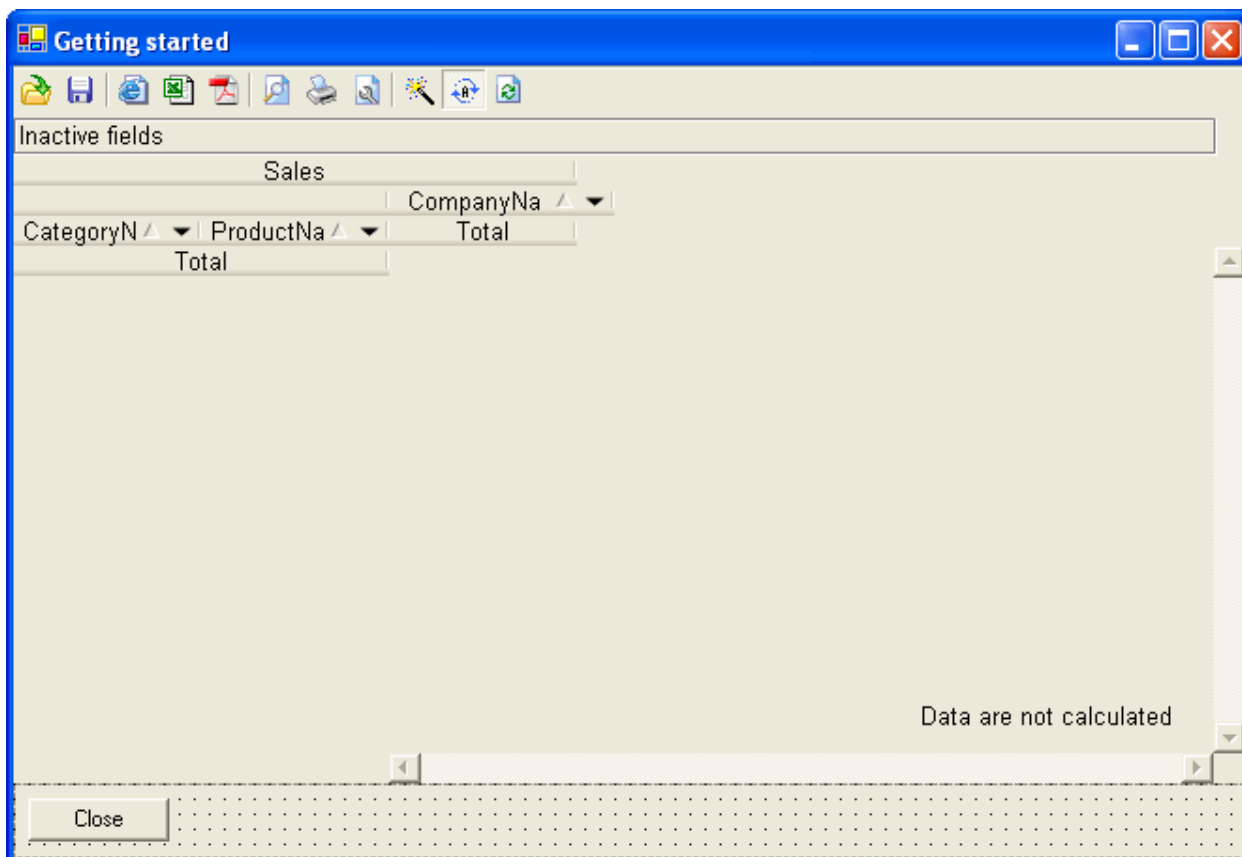
On the Style sheet tab specify a presentation style of your pivot table. It is possible to select one of the existing styles by clicking the "Auto format" button.



Add the DataCubeGrid object onto the form.



Indicate the adjusted DataCube object as the DataCubeGrid Source property value.



Add the code for filling the Sales table from the DataSet into the form uploading event handler.



Run your application and in its form you will see the pivot table created by you.

Inactive fields		Sales						
CategoryN	ProductNa	CompanyNa	Bigfoot Breweri	Cooperativa de	Exotic Liquids	Forks d'irable	Grandma Kelly'	Total
Beverages	Sasquatch Ale		6350,40					6350,40
	Steeleye Stout		13644,00					13644,00
	Total		22391,20		29144,06			51535,26
Condiments	Aniseed Syrup				3044,00			3044,00
	Grandma's Boy						7137,00	7137,00
	Northwoods Cr						12772,00	12772,00
	Siroop d'irable					14352,60		14352,60
	Total				3044,00	14352,60	19909,00	37305,60
Confections	Tarte au sucre					47234,97		47234,97
	Total					47234,97		47234,97
Dairy Produ	Queso Cabrale			12901,77				12901,77
	Queso Manche			12257,66				12257,66
	Total			25159,43				25159,43
Produce	Uncle Bob's Or						22044,30	22044,30
	Total						22044,30	22044,30
Total			22391,20	25159,43	32188,06	61587,57	41953,30	183279,56

In this table you will be able to change the layout by dragging field headers. You will also be able to assign sorting and filtering by value and condition.



## Basic Information

### Concept

SharpShooter OLAP pivot table is a kind of an interactive table designed to be used for statistical analysis of multiple data. The rows and columns of such interactive table are based on the data taken from several columns of an initial table or another data source.

The information in pivot table cells is the data aggregated by the corresponding row and column.

Generally, an OLAP cube represents a structure containing multidimensional OLAP data, i.e. dimensions – the descriptive data that form the axes of a multidimensional cube, and facts – the computational quantitative values. Dimensions can contain multilevel hierarchies of values, and measures are aggregated data (sums, mean, maximal or minimal value, number of records) based on the data source fields that are of interest in respect to statistical analysis.

The formation of a pivot table presupposes the execution of the following operations:

- Data grouping;
- Calculation of sub-group subtotals;
- Calculation of totals.

The field objects are used to define dimensions and facts. In a field, one can specify the initial table data to be used for a pivot table calculation and the way this pivot table will be provided to a user. The field indication order for one dimension also assigns the grouping of data within this dimension.

A user is the one who determines the assigning of field resultant values (for both dimensions and facts). It can be implemented by means of assigning an expression to a field. An aggregation function is assigned for facts. Thus, the resultant value for facts is the aggregation of their expressions results. So, the process of pivot table formation is being conducted as follows:

- At the data grouping stage the tree structure of table dimensions is being formed. For each tree node (dimension element value) the calculation of a corresponding field expression in respect to the data source information takes place. Here the grouping is being taken into account: group – tree node (a corresponding field from the dimension element list is being inserted), sub-group – the leaves of this tree node (the following field in the dimension elements list).
- The calculation of sub-group subtotals stage is the stage at which the direct filling of a resultant pivot table is being conducted. Fact values for dimension intersections are being calculated through corresponding fields' expressions. The calculation of expressions is performed subject to the values of all dimension groups formed at the data grouping stage.
- The calculation of totals occurs concurrently with filling a table for sub-groups in general. It is being held taking into account the aggregate functions specified in fields on the basis of fact values already calculated.

In addition, a user can alter the sorting and conduct a filtering by random data combination, display data as percentage and perform various useful conversions of a pivot table.

### Data Sources

The following classes can be used as data sources for SharpShooter OLAP:

- ADO.NET objects – System.Data.DataSet, System.Data.DataView, System.Data.DataTable;
- user-defined classes (Business Objects) that implement System.ComponentModel.IListSource or System.Collections.Ilist interface.

Data sources for pivot tables are specified through the DataSources property of the DataCube class. This property is of System.Object type.





In order to specify a data source for a pivot table via the property editor, select the DataCube object and activate the DataSource property editor in the Properties window. There will appear a list containing the names of objects that are present on the form and fit to be used as data sources. It is necessary to select the name of an object intended to be used as a data source.

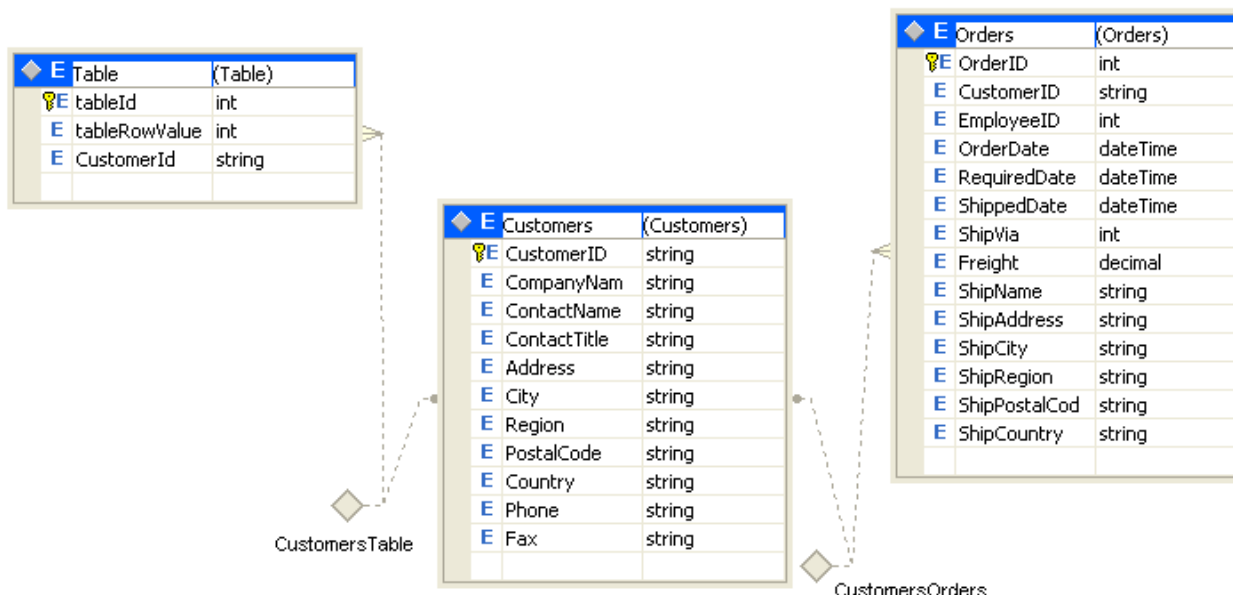
After that, the properties of an object selected as a data source will be available in the DataCube object for pivot table adjustment.

Nested data sources.

SharpShooter OLAP supports nested data sources. Now there is no need to prepare a special data access meeting the required calculations. It is enough to simply upload data from several connected tables into a single DataSet. Then it is necessary to specify a parent table to the DataCube object as a DataSource property. The JointFields property (string array) is filled in such a way that the property contains a list of data source fields through which a connection with the data source child elements is executed.

For example, in case one uses data schemas displayed in the screenshot below, the Customers table should be specified to the DataCube object's DataSource property and the JointFields set should be filled in the following way:

"CustomersTable"  
"CustomersOrders"



In dimension and fact elements' expressions it is possible to refer to nested data source fields specifying a path to the source. E.g., "CustomersTable.tableRowValue" or "CustomersOrders.OrderId"

### Working with the DataCubeGrid Component (Pivot Table Presentation, Manipulation)

The DataCubeGrid object is an interactive control which displays a pivot table on the application form. There a user can view a final pivot table, modify its appearance and content, print or export a table into another file format.

The image below shows the appearance of DataCubeGrid.

The screenshot shows a DataCubeGrid interface with a pivot table. The interface includes a toolbar at the top, followed by filters for Quarter, Month, and Inactive fields. Below these are filters for Year (1996) and CompanyName (Tokyo Traders, Zaanse Snoepfabriek, Total). The pivot table has columns for Discount, Facts fields (Sales sum, Discount), and Disc. The rows are categorized by ProductName, with sub-headers for Condiments, Confections, Dairy Products, Grains/Cereals, and Meat/Poultry. Annotations include: 'Y dimension fields' pointing to ProductName, 'Sorting indicator' pointing to a triangle icon, 'Filter button' pointing to a small square icon, and 'Collapse buttons' pointing to plus and minus icons.

Category	Product Name	Discount	Facts fields (Sales sum)	Discount	Sales sum	Disc
Condiments	Original Frankfurter grüne				3950,05	0,4
	Sivon d'érable				5831,10	0,5
	Vegete-spread				7770,30	0,0
	Total				38602,31	3,1
Confections	Total		1301,75	0,4131%	63792,07	5,3
Dairy Products	Total				87984,04	4,8
Grains/Cereals	Total				34607,19	2,8
Meat/Poultry	Alice Mutton				11612,25	0,5
	Mishi Kobe Niku	0,0000%			291,00	0,0
	Perth Pasties				4808,48	0,4
	Petit chinois				6740,40	0,7
	Thüringer Rostbratwurst				41766,75	1,3
	Tourtière				781,88	0,9
	Total	0,0000%				66000,75

The `Inactive fields` scope displays headers of the fields that are specified in DataCube, but not engaged in pivot table formation.

The `X dimension fields` scope displays headers of the fields that define pivot table columns.

The `Y dimension fields` scope displays headers of the fields that define pivot table rows.

The `Fact fields` scope contains headers of the fields that are the facts of a pivot table.


The fields in `X dimension fields` scope and in `Y dimension fields` scope correspondingly specify columns and rows of a pivot table. The width of any `DataCubeGrid` column can be altered by dragging its borders with a mouse.

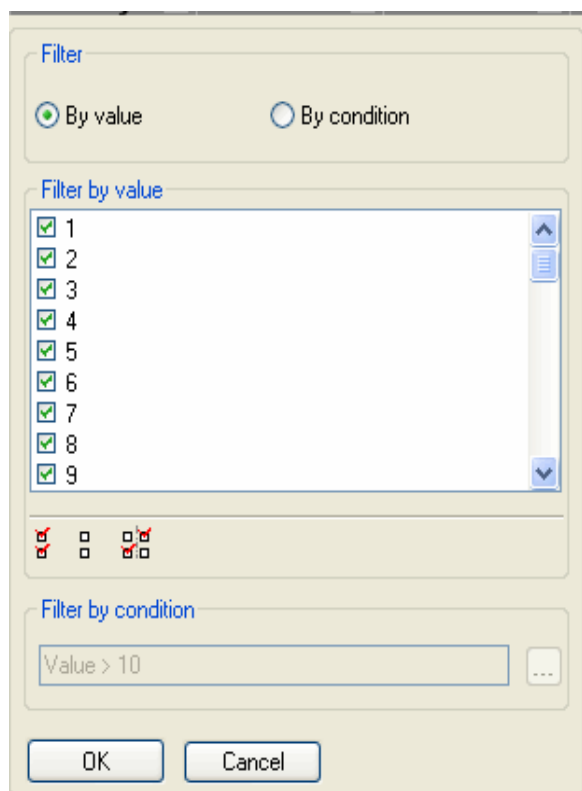
The screenshot shows a DataCubeGrid interface with a pivot table. The interface includes a toolbar at the top, followed by filters for ProductName, Year (1994), Month (August), and Quarter (3). The pivot table has columns for Sales sum and Discount, grouped by Year, Month, and Quarter. The rows are categorized by Category. A mouse cursor is shown over the 'Sales sum' column header for the 'August' group.

Category	Sales sum	Discount	Sales sum	Discount	Sales sum
Beverages	3182.50	0.6196%	3182.50	0.6196%	4866.88
Condiments	1878.20	0.4544%	1878.20	0.4544%	2296.60
Confections	5775.15	0.4544%	5775.15	0.4544%	5006.77
Dairy Products	6838.34	0.4544%	6838.34	0.4544%	4761.70
Grains/Cereals	1256.86	0.0826%	1256.86	0.0826%	270.60
Meat/Poultry	2661.72	0.1239%	2661.72	0.1239%	3495.66
Produce	3868.80	0.3305%	3868.80	0.3305%	804.32
Seafood	2400.32	0.5370%	2400.32	0.5370%	3982.74
Total	27861.89	3.0568%	27861.89	3.0568%	25485.27




Directly below the dimension field headers, the values of these dimensions (subject to their grouping) are displayed. A grouping is defined by the DataCube fields indication order. It is also possible to drill down/roll up group values ([-] / [+] buttons) on the stipulation that this group contains sub-groups with values different from the Total.

It is possible to drag field headers. They can be moved between Inactive fields, X dimension fields, Y dimension fields and Facts fields scopes. It is also possible to change their order within one scope.

The field headers are interactive. Each of them has a filter button (  ), which helps a user to enter a filter setup dialogue for a respective field



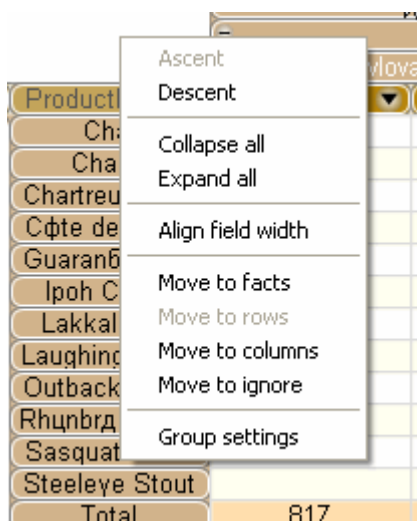
There are two possible ways of filtering field values: filtering by value and filtering by condition.

The setting of filtering by value presumes selection of those field values which will be displayed. It is also possible to set a filter for all field values by means of clicking the "Select all" button (  ). To cancel filtering a user should click the "Unselect all" button (  ). To invert filter selection a user can click the "Invert" button (  ).

To filter data by condition it is necessary to specify an expression, which result would satisfy the desired field value selection. A press of the "OK" button leads to saving filter condition and pivot table recalculation subject to the preset filtering. The "Cancel" button leads to ignoring all filter alterations and closing the dialogue window.

Field headers used in pivot table calculation allow specifying pivot table sorting. When the field header of the dimension is clicked the sorting by this dimension is carried out taking into account the grouping. When the field header that defines table fact is clicked, the sorting by the fact values column is conducted taking into account its dimensions. The sequence of displaying strings is changed. This capability of sorting by total columns is intended to facilitate quantitative data analysis.

When the right mouse button is pressed, the following contextual menu appears.



The contextual menu contains the following items:

Ascent – is used to set ascending sorting;

Descent – is used to set descending sorting;

Collapse all – is used to roll all field groups in;

Expand all – is used to roll all field groups out;

Align field width – is used to align field width by maximal width of its value;

Move to facts – is used to move a field into facts;

Move to rows – is used to move a field into row dimensions;


Move to columns – is used to move a field into column dimensions;


Move to ignore – is used to ungroup a field from pivot table calculation.


Group settings – is used to open the window for the field groups manipulation.


The DataCubeGrid tool set also contains ToolBox.


It has the following buttons:

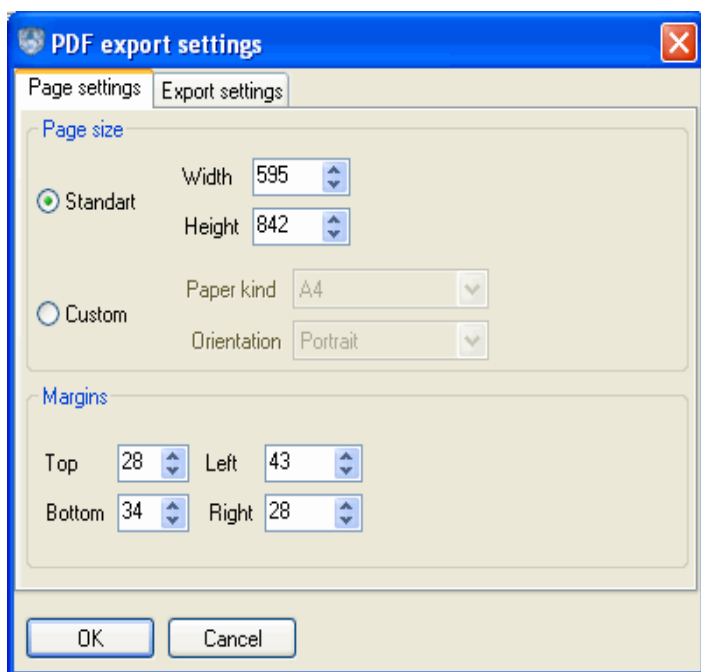
The "Open" button (). A press on this button allows a user to select a file, from which the DataCube layout and/or presentation style will be downloaded.

The "Save" button (). A press on this button calls the file saving dialogue. Here one can indicate what should be saved: a DataCube layout, a pivot table presentation style or both.

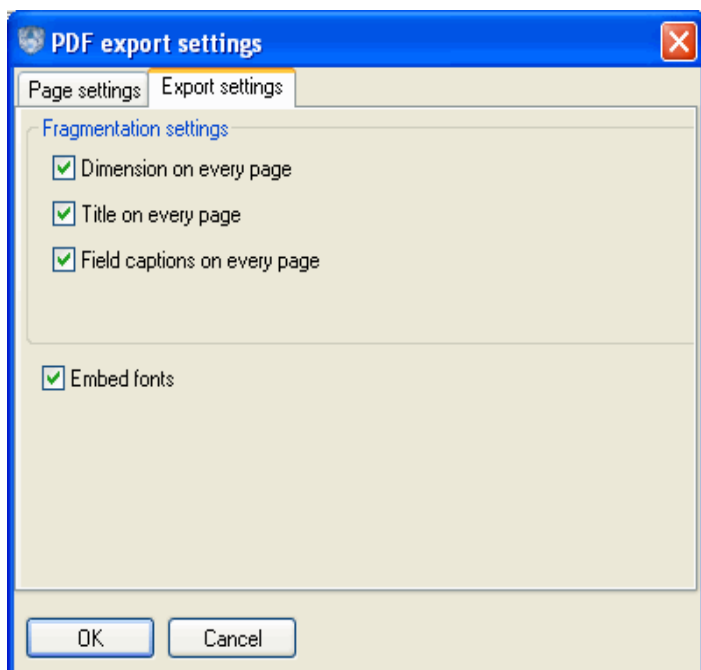
The "Export to HTML" button (). A press on this button will allow a user to enter the export to HTML dialogue box. At that, the current state of a pivot table (layout, styles, column width, sorting, and filters, the dimension element groups collapse) is being taken into account.


The "Export to Excel" button (). A press on this button opens the export to MS Excel dialogue box. A file is saved including the current state of a pivot table.


The "Export to PDF" button (). A press on this button opens the PDF export settings dialogue box. On the Page Settings tab a user is able to specify final PDF document settings: page size, orientation and indents.




On the Export Settings tab a user will see the pivot table appearance settings in a PDF document. Here, a user can specify whether the pivot table dimensions, its header and fields' names for dimensions and facts will be repeated on each document page. In addition, a user can specify whether to include the fonts used in a pivot table into the document.




The "Print setting" button (  ). This button opens the print settings dialogue.


The "Print preview" button (  ). This button allows previewing a pivot table before printing. At that, the settings specified in the Print setting dialogue window are retained and a pivot table is split into pages.

The "Print button" (  ). This button allows printing a pivot table.

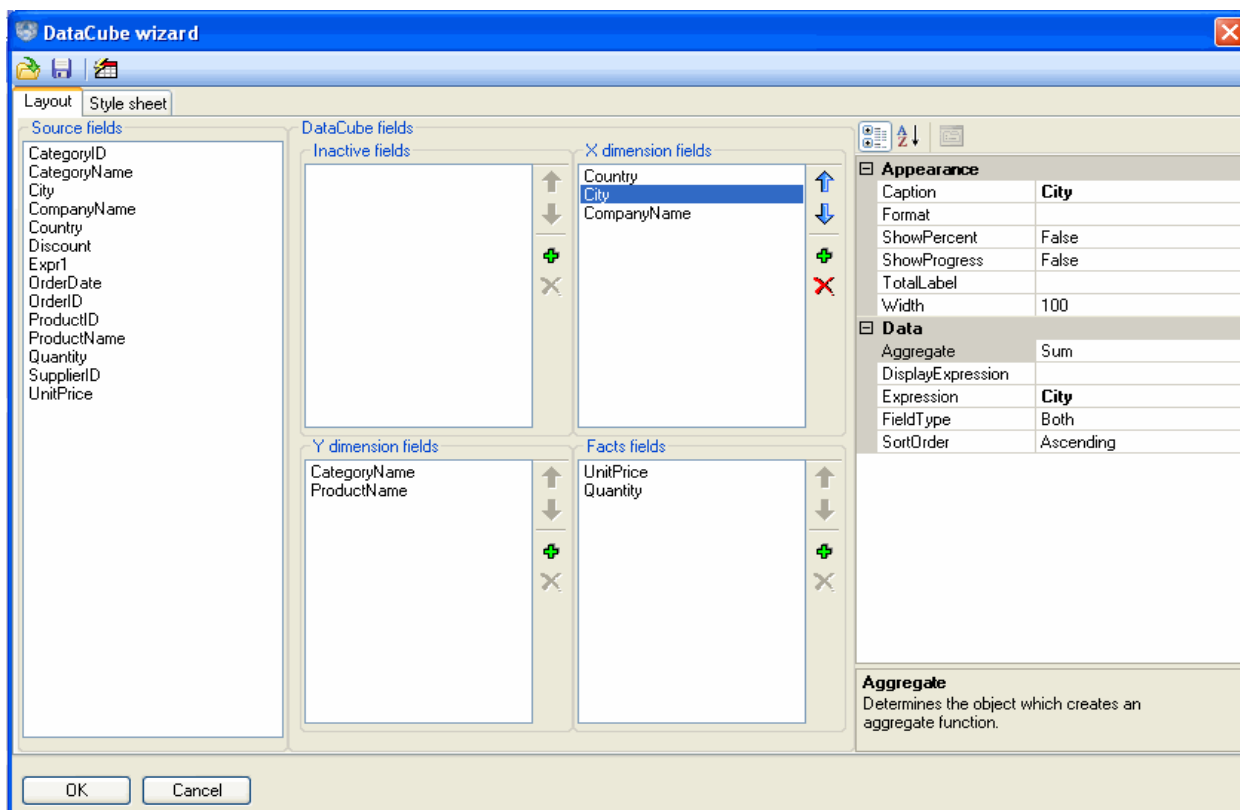
The "Recalculate" button (  ). A press on this button calls forced recalculation of a pivot table.



The "Auto recalculate" button (  ). A press on this button turns on/off pivot table automatic recalculation while changing table layout and filtering conditions. This button usage is reasonable while working with large body of data. In order to recalculate a pivot table after specifying the settings, you should use the Recalculate button together with the Auto recalculate button pressed.

The "Wizard" button (  ). When pressed, this button runs the DataCube wizard for pivot table retrofitting.

The "DataCube wizard appears as follows.



The DataCube wizard allows altering a pivot table layout and a style of its display. The DataCube wizard provides four lists containing fields for pivot table formation:

Source fields – the list that provides a user with all data source fields;


Inactive fields – the list of fields selected by a user for pivot table formation, but not used for pivot table calculation;

X dimension fields – the list of fields selected from a data source or created by a user for column dimension formation;


Y dimension fields – the list of fields selected from a data source or created by a user for row dimension formation;


Facts fields – the list of fields, selected from a data source or created by a user for pivot table facts formation.

A user is capable of dragging fields from one list into another or within one and the same list, thereby changing the order of fields. In order to make pivot table field lists contents alteration a more convenient process, each list has a button panel.

The "Move up" button (  ) allows moving a selected field one position up within a list;



The "Move down button" () allows moving a selected field one position down within a list.

When pressed, the "Add field" button () creates a new field which is added in a list.

The "Remove field" button () allows removing selected fields from a list.

The DataCube wizard window contains a selected field property editor (Fields properties).

Appearance	
Caption	Month
Format	
ShowPercent	False
ShowProgress	False
TotalLabel	
Width	100

Data	
Aggregate	Sum
DisplayExpression	
Expression	getMonth (OrderDate)
FieldType	DimensionOnly
SortOrder	Ascending

**FieldType**  
Determines the mode representing a way the given field can be used.

The Expression property assigns an expression for a field;

The Caption property assigns an inscription on a field caption;

The Aggregate property assigns a data aggregation type for a field (makes sense for facts only);

The Field type property assigns a field type (Fact only, Dimension only, Both);

The Format property assigns a field data output format string;

The Width property assigns a field width;

The Show percent property defines, whether a field data will be presented as percentage;

The Show progress property defines whether a cell will be presented as progress. To set this property, it is necessary to set the Show percent property (makes sense for facts only);


The Total label property assigns a header for a field total;


The Sort order property assigns a field data sorting order.

The DataCube wizard also has the following buttons:

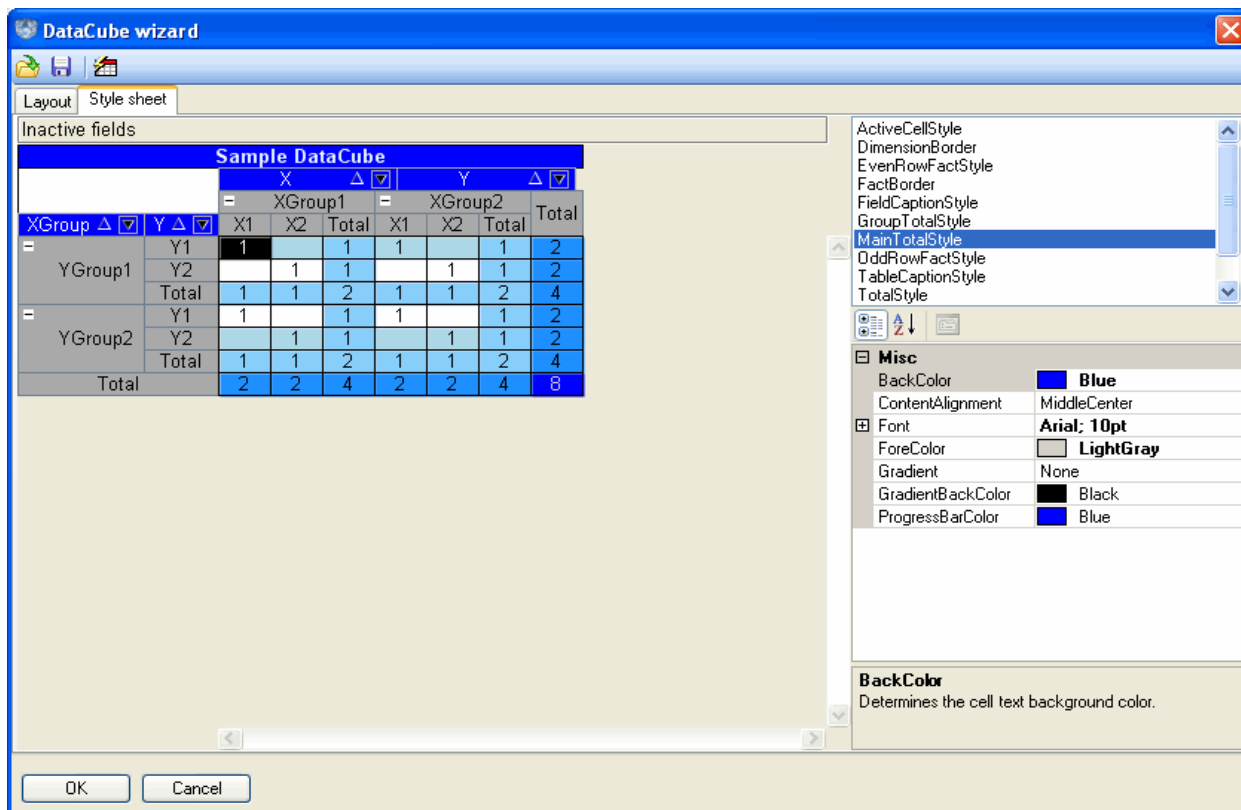
The "Open" button () ; it allows uploading pivot table settings.



The "Save" button (  ); it allows saving pivot table settings. It is possible to save a pivot table layout, its presentation style or both.

The "Auto format" button (  ); it opens a style selection dialogue box.

DataCube wizard has the Style sheet tab.



Here a user can specify pivot table appearance settings. In particular, a user can define displaying styles for X and Y dimensions, fields' headers, even/odd strings of facts table, totals crossings etc. In order to do this one should select a table element from the list and specify the corresponding properties to this element in the Property grid.





## Expressions

### Destination and General Principles of Expression Use

It is possible to assign an expression for all fields specified in a pivot table (facts and dimensions). The expression calculation result defines a field value, and therefore the dimension elements values and fact values. As for facts, values defined by their expression results take part in aggregation.

The order of expression use is as follows: during a pivot table formation a field responsible for a dimension element or a fact is assigned for dimension element or fact cell value calculation. There takes place a calculation of such field expression taking into consideration the dimension intersection for facts and the data source value for dimension elements. The calculated value is recorded in a table.

If an expression contains syntax or semantic errors or an exception appears during the calculation process, the component will generate a corresponding exception message and provide a user with information on the error type and location.

### Description of the Expression Language Syntax

An expression is a task on formula calculation. The expression value is a calculation result. The expression is built out of constants, objects and their properties and methods, operators, function calls and round brackets.

Data types used:

In the expressions all data types, available in .NET Framework, are used. Also there is the special support for fractional and logical values, strings and the DateTime type.

The following operations are available in expressions:

Arithmetic operations:

- `\+'` - addition. It is defined for numerical values;
- `\-` - subtraction. It is defined for numerical values;
- `\-'` - unary minus. It is defined for numerical values;
- `\*'` - multiplication. It is defined for numerical values;
- `\/'` - division. It is defined for numerical values.
- `\%'` - residue of division. It is defined for numerical values.

Logical operations:

- `\or'` - logical OR. It is defined for logical values;
- `\and'` - logical AND. It is defined for logical values;
- `\='` - equality. It is defined for numerical, string and logical values;
- `\! ='` - inequality. It is defined for numerical, string and logical values;
- `\<'` - less. It is defined for numerical values;
- `\<='` - less or equal. It is defined for numerical values;
- `\>'` - more. It is defined for numerical values;
- `\>='` - more or equal. It is defined for numerical values;



'not' - logical negation. It is defined for logical values.

String operations:

'&' - strings concatenation. It is defined for string values.

The descending priority operations order.

'-(unary), 'not'

'\*', '/', '%'

'+', '-', '&'

'>', '>=', '<', '<='

'=', '!='

'and'

'or'

Constants assigning:

< decimal numeral >\* - the integral value.

< decimal numeral >\*.< decimal numeral >\* - value.

For example, 1234 is an integral value, 1234.1234 is a fractional value.

String constants are assigned with the help of a double quotation mark (")

For example, "It is a string value".

Logical constants are assigned with the help of two key words: 'true' and 'false'.

'Null' constant. This constant is used for indeterminate value assigning.

Access to variables, objects, their fields and methods.

In the expressions one can apply to variables, available in a field by their names. The search for variables and objects by their names is executed in the following way: in case an address by the object's name is present in an expression, the GetObject function that should return the desired object is called for a field the given expression is written for. A field as an object is available by "value" name.

A dot '.' is used to apply to object fields and methods.

For example:

Value.Year - a reference to the field Year property the value of which is of the DateTime type;

Field1.ValueToString() - a call for the field Field1 ValueToString() method;

ATTENTION! It is impossible to call the object methods, if they are overloaded.

As the expression language is not typified, it is impossible to define what method should be called.

In the expressions, one can apply to the functions available in a field by their names. The search is executed similarly to the objects search by the names.

The following functions are available:



The functions of conditional choice:

`if (condition, value1, value2)`, the 1st parameter should be of logical type; the 2nd and the 3rd parameters should be of object type. If the 1st parameter is true, the function returns the 2nd parameter value, otherwise it returns the 3rd parameter value.

`switch (condition[0], value[0], ... , value by default)`, if the [i] condition is set to true, the function returns the [i] value, if the [i] condition in any i is false, the value returns by default.

#### Formatting Functions:

`format (object, mask)`, the object is of object type, the mask is of string type. It returns the object's string presentation according to the mask.

#### Mathematical Functions

`sin (argument)` – calculates the argument sine. The argument is of the double type, the calculation result is of the double type.

`cos (argument)` – calculates the argument cosine. The argument is of the double type, the calculation result is of the double type.

`tan (argument)` – calculates the argument tangent. The argument is of the double type, the calculation result is of the double type.

`atan (argument)` – calculates the argument arctangent. The argument is of the double type, the calculation result is of the double type.

`sqr (argument)` – calculates the argument to square. The argument is of the double type, the calculation result is of the double type.

`sqrt (argument)` – calculates the argument's square root. The argument is of the double type, the calculation result is of the double type.

`log (argument)` – calculates the argument's natural logarithm. The argument is of the double type, the calculation result is of the double type.

`exp (argument)` – raise E number to the argument degree. The argument is of the double type, the calculation result is of the double type.

`sign (argument)` – returns: -1 if the argument is negative, 0 – if the argument is 0, 1 – if the argument is positive. The argument is of the double type, the calculation result is of the int type.

`abs (argument)` – returns the argument's absolute value. The argument is of the double type, the calculation result is of the double type.

`round (argument1, argument2)` – rounds the 1st argument value to the symbols amount after the comma, assigned by the 2nd argument. The 1st argument is of the double type, the 2nd argument is of the int type, the calculation result is of the double type.

The expression can be the argument of any function and method. If the expression calculation result is not appropriate to the argument type, the expression will not be calculated.

#### Strings Processing Functions:

`concat (string1, string2)` – returns concatenation of two strings assigned by parameters;

`indexOf (string1, string2[, start])` – returns index of occurrence of the substring string2 to the string1 string from the Start position;

`insert (string, position, subString)` – pastes subString into the String from the Position;



`length(string1)` – returns `string1` length;

`padLeft(string, length, char)` – returns `string` complemented by the `char` character on the left up to the specified length;

`padRight(string, length, char)` – returns `string` complemented by the `char` character on the right up to the specified length;

`remove(string1, start, length)` – removes the `length` substring from `string1` beginning from the **Start** position;

`replace(string, subString, newSubString)` – replaces substring with `newSubString` in the `string`;

`substring(subString (string, start, length))` – returns the `string` substring with the specified length beginning from the **Start** position;

`toLowerCase(string)` – casts `string` to the lower registry;

`toUpperCase(string)` – casts `string` to the upper registry;

`trim(string)` – removes white space characters from the `string`;

`trimEnd(string)` – removes white space characters from the end of the `string`;

`trimStart(string)` – removes white space characters from the beginning of the `string`;

`regIsMatch(string, expressionString)` – returns match char of the `string` to the `expressionString` regular expression;

`regReplace(string, expressionString, newString)` – replaces substring, corresponding to the `expressionString` regular expression, with `newString` in the `string`;

#### Date Processing Functions:

`getMonth (argument)` – returns the `string` month name by the date assigned by an argument. An argument should be of the `System.DateTime` type.

`getDayOfWeek (argument)` – returns the `string` day of week name by the date assigned by an argument. An argument should be of the `System.DateTime` type.

`getQuarter (argument)` – returns the quarter number of a date specified as an argument. An argument should be of the `System.DateTime` type.

`addDays(date, n)` – adds `N` number value of days to the `Date` object;

`addHours(date, n)` – adds `N` number of hours value to the `Date` object;

`addMinutes(date, n)` – adds `N` number of minutes value to the `Date` object;

`addMonths(date, n)` – adds `N` number of months value to the `Date` object;

`addSeconds(date, n)` – adds `N` number of seconds value to the `Date` object;

`addYears(date, n)` – adds `N` number of years value to the `Date` object;

`getDate(date)` – returns the `Date` object without time;

`getDayOfMonth(date)` – returns the day of the month value of the `Date` object;

`getDayOfYear(date)` – returns the day of the year value of the `Date` object;

- getHour (date) – returns the hours of the specified date value of the Date object;
- getMinute (date) – returns the minutes of the specified date value of the Date object;
- getSecond (date) – returns the seconds of the specified date value of the Date object;
- getTime (date) – returns the time of the specified date value of the Date object;
- getYear (date) – returns the year of the specified date value of the Date object;
- toShortDateString (date) – returns string representation of the date;
- toShortTimeString (date) – returns string representation of the time.

The ranking function:

Range (currentValue, range, zeroPoint) – extracts data intervals. All arguments should be of numeral type. currentValue – the current value; range – the range; zeroPoint – the zero point.

Example:

Let us create a new field and specify "Range (OrderDate.Year, 4, 0)" as the Expression property value. Here the OrderDate is a System.DateTime type data source field. The result is presented in the image below.

OrderDate	Quarter	Month	Quantity			
<b>Purchases</b>						
	RangeField	Year				
	=	1992 - 1996	= 1996 - 2000			
Category	1994	1995	Total	1996	Total	Total
+ Beverages	1693.40	4587.00	6280.40	5531.25	5531.25	11811.65
+ Condiments	626.30	2077.20	2703.50	1901.80	1901.80	4605.30
+ Confections	1047.90	3388.50	4436.40	3112.90	3112.90	7549.30
+ Dairy Produ	1435.80	5006.20	6442.00	3433.80	3433.80	9875.80
+ Grains/Cere	422.40	2192.15	2614.55	1549.75	1549.75	4164.30
+ Meat/Poultr	973.40	3450.42	4423.82	2993.51	2993.51	7417.33
+ Produce	526.20	2216.10	2742.30	2044.15	2044.15	4786.45
+ Seafood	755.50	3100.04	3855.54	2435.24	2435.24	6290.78
Total	7480.90	26017.61	33498.51	23002.40	23002.40	<b>56500.91</b>

Realization of custom functions and variables is available in the expressions.

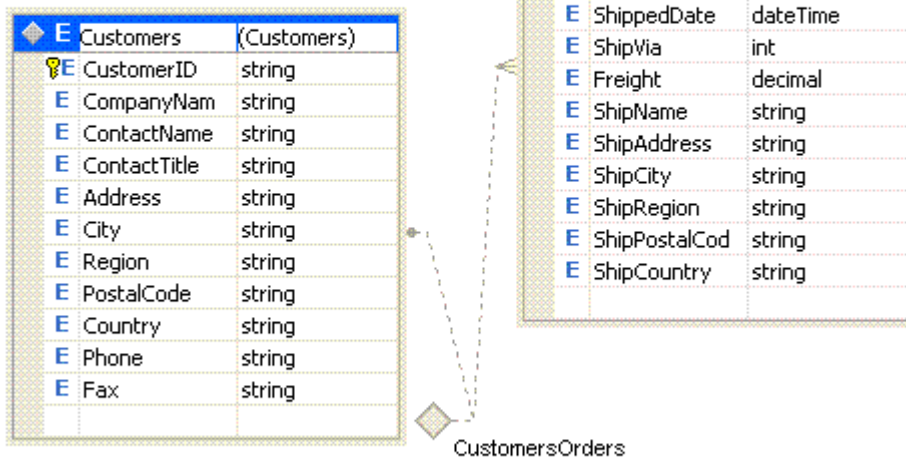
To make custom functions and variables available in the elements expressions it is necessary to implement custom element that will realize the PerpetuumSoft.Framework.Expressions.IExpressionSite interface. After that it is necessary to register custom function. The registration is carried out the following way:

```
PerpetuumSoft.Framework.Expressions.BuiltInFunctions.RegisterFunction(<Function name>, <implemented IFunction object>);
```

Access to the nested fields of a data source.

In expressions it is possible to refer to the data source nested fields. The reference itself is executed with specification (specifying) of the complete path to a field relative to the main source specified as a value to the DataCube.DataSource property. E.g., in case a data schema displayed in the screenshot below is used the access code for the Orders table's OrderDate field and Freight field should look in the following way:

"CustomersOrders.OrderDate" and "CustomersOrders.Freight", correspondingly.








## Work in Windows Forms Applications


### Components Used in Windows Forms Applications

The four components used for displaying and managing pivot table in Windows Forms applications are the DataCube, DataCubeGrid, DataCubeView and DataCubePrintDocument.

The DataCube (  ) component serves for organizing the data from an outer source into a pivot table. DataCube stores pivot table layouts, its presentation style and allows pivot table print and export.

The DataCubeGrid (  ) component is intended for displaying the data presented in the DataCube. The DataCubeGrid enables a user to interactively shift pivot table layout and presentation style as well as to manage its print and export.

The DataCubePrintDocument (  ) component is responsible for splitting a pivot table into pages and leading it to a printing device.

The DataCubeView (  ) component allows creating presentations of data stored within the DataCube object. It is necessary to provide an interface for access to data stored in the DataCube by other components that use the same information (like chart and diagram components). The DataCubeView component "expands" the pivot table dimension cube providing the data as a linear table, which columns are facts of the initial cube and a field of one of dimensions. DataCubeView allows reorganizing a cube into a linear table by any dimension.

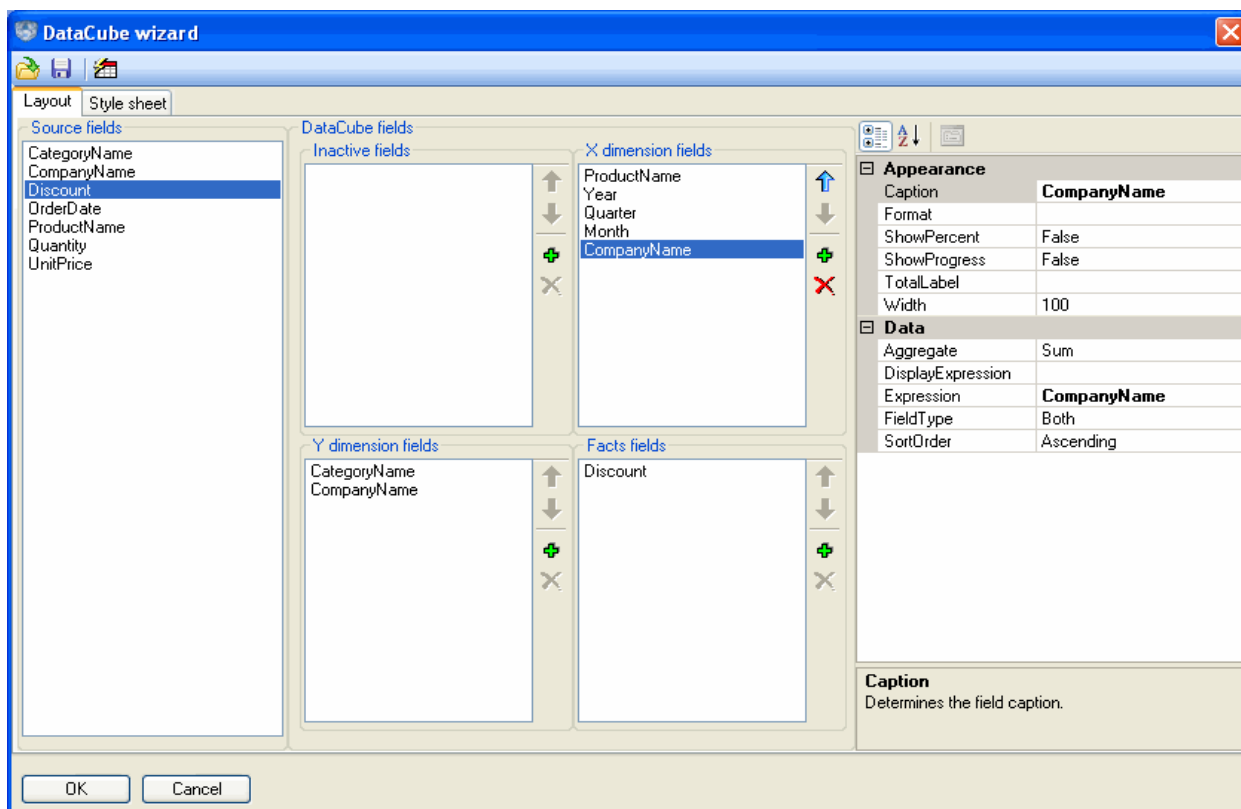
### The Use and Capabilities of the DataCube

Let us examine the use of the DataCube component by the following example.

Let us create a Windows Forms application and add the DataCube component object situated on the SharpShooter OLAP tab onto the form. Let us name the newly-created object salesDataCube. It is necessary to specify a data source (the DataSource property). To do that, we shall add the DataSet object onto the form and create a Sales table for this object. The Sales table structure is shown in the image below.

Field name	Type	Description
CategoryName	String	Name of food category
ProductName	String	Name of product
CompanyName	String	Name of company
UnitPrice	Currency	Unit price
Quantity	Number	Quantity
Discount	Number	Discount
OrderDate	DateTime	Date fo sale

In order to adjust the desired pivot table layout let us call the DataCube wizard designer by double clicking the salesDataCube object or by selecting the "Run Designer" option in the salesDataCube property window.



The designer Source fields list contains the Sales table field names. Let us assign a pivot table row dimension. To do that, we shall drag the CategoryName and ProductName fields from the Source fields list into the Y dimension field list (category and product names). This will be a row dimension, in which the products are grouped by name and category. Now we shall assign column dimension. Let it be the date of sale indicating year, month, day of week and the name of a producer company. Let us create three fields within the X dimensions fields list and set the Expression property for the newly-created fields:

```
Year      - OrderDate.Year
Quarter   - getQuarter(OrderDate)
Month     - getMonth(OrderDate)
```

Also we shall drag the CompanyName field from the Source fields list into the X dimension fields list. Thus we have created a column dimension. Here the data will be grouped by company name and by date identifying year, quarter, and month.

Let us specify the Field type property for all dimension fields as Dimension only.

Let us specify the facts for our pivot table. First of all, we shall create a field in the Facts fields list and name a new field by specifying the "Sales sum" as the Caption property value. We shall indicate "UnitPrice\*Quantity\*(1 - Discount)" as the Expression property value for this field. This fact will represent the final sales total for a product of a specific company on a specific date taking in consideration the unit price, the amount sold and discounts. We shall also drag the Discount field from the Source fields list into the Facts fields list. Then let us set the Show percent and Show progress properties to true. And after that, let us select an aggregate function for the given fact. Specify the Avg value for the Aggregate property. The given fact will indicate the mean discount value for the given product as percentage. Let us also set the Format property to "0.00" for all fields from the Facts fields list.

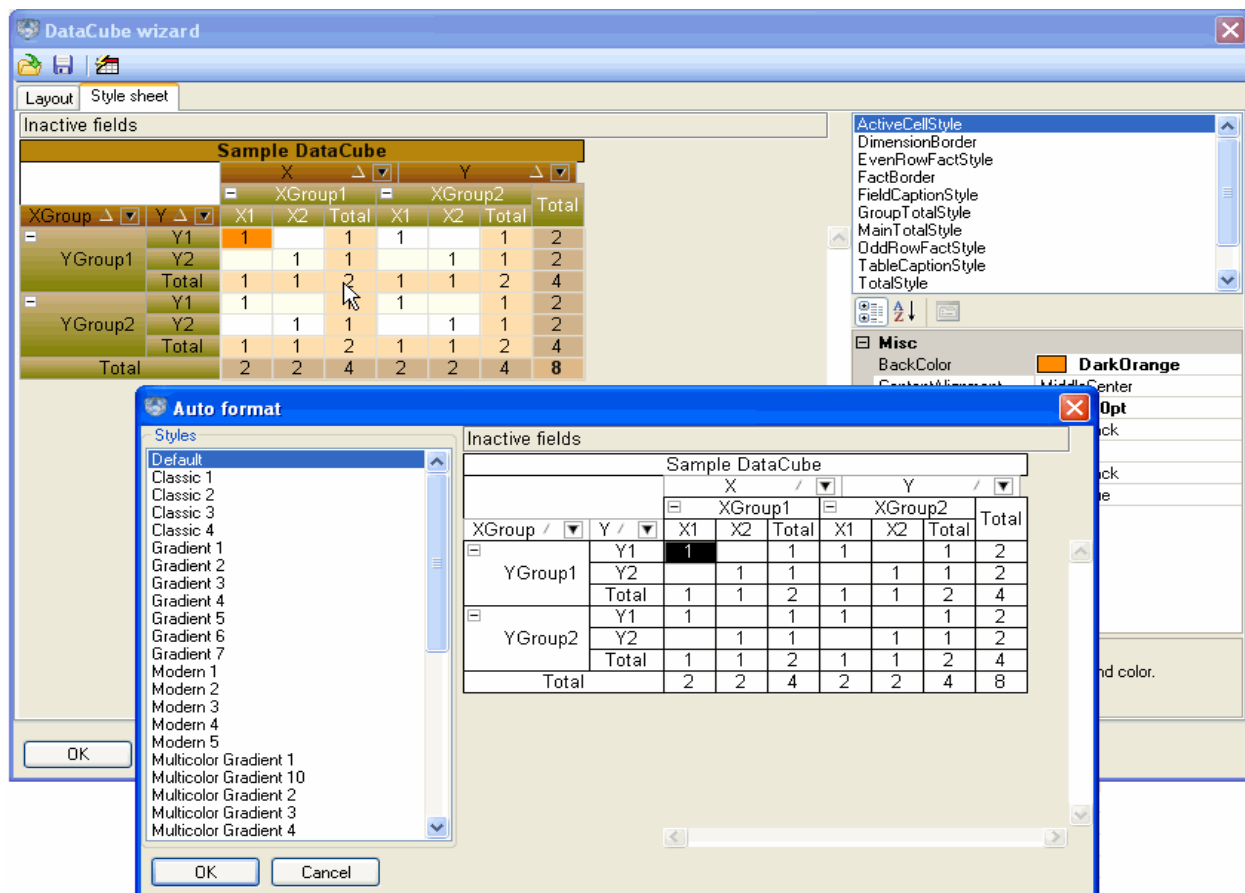
The pivot table layout is now assigned.





		Year	Quarter	Month	CompanyName						
		Year1									
		Quarter1				Quarter2					
		Month1		Month2		Month3					
CategoryName	ProductName	Company1		Company2		Company3					
Category1	Product1	SalesSum11	Discount11	SalesSum12	Discount12	SalesSum13	Discount13	SalesSum14	Discount14		
	Product2	SalesSum21	Discount21	SalesSum22	Discount22	SalesSum23	Discount23	SalesSum24	Discount24		
Category2	Product3	SalesSum31	Discount31	SalesSum32	Discount32	SalesSum33	Discount33	SalesSum34	Discount34		
	Product4	SalesSum41	Discount41	SalesSum42	Discount42	SalesSum43	Discount43	SalesSum44	Discount44		

Now let us assign the pivot table presentation style in the Style sheet designer tab.



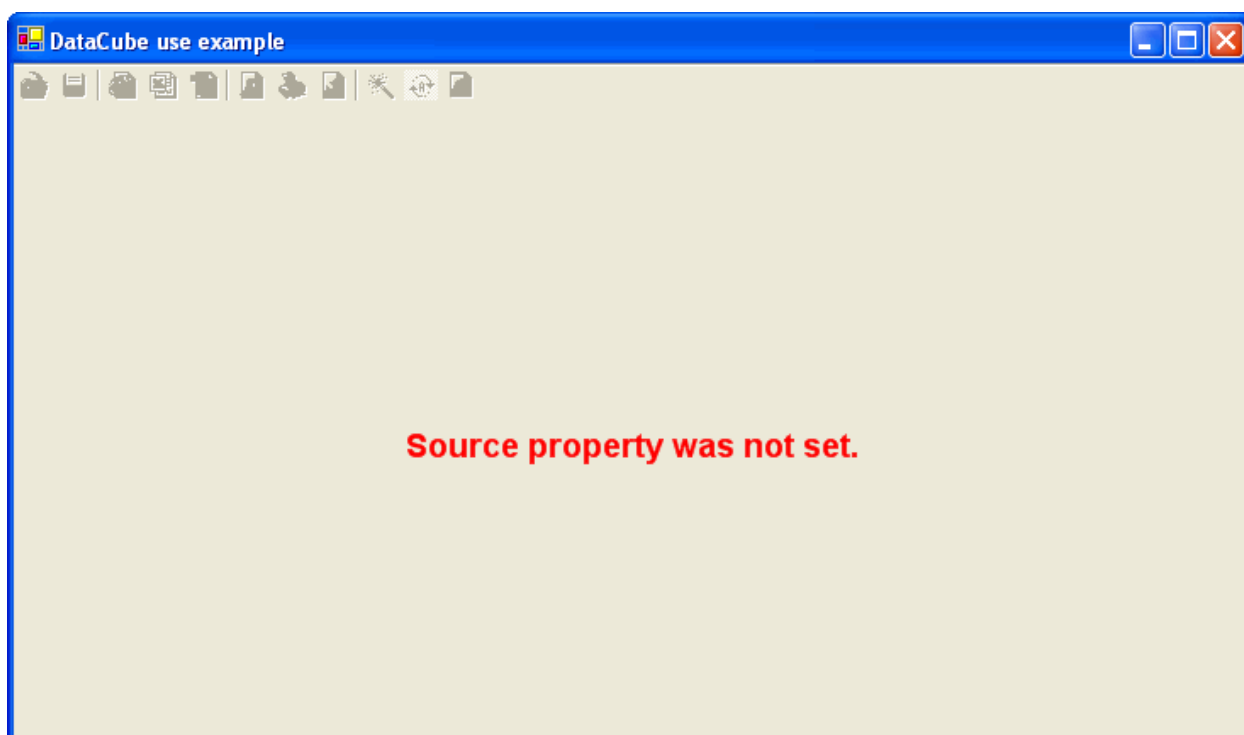
To do that, we shall launch the auto format call dialogue and select one of the predefined pivot table presentation styles. Then we shall save the pivot table settings having specified the "Layout and sheet" file type in the saving dialogue box.

You can find this example in the DataCubeUseExample folder.

### The Use and Capabilities of the DataCubeGrid

We shall demonstrate some DataCubeGrid capabilities by an example.

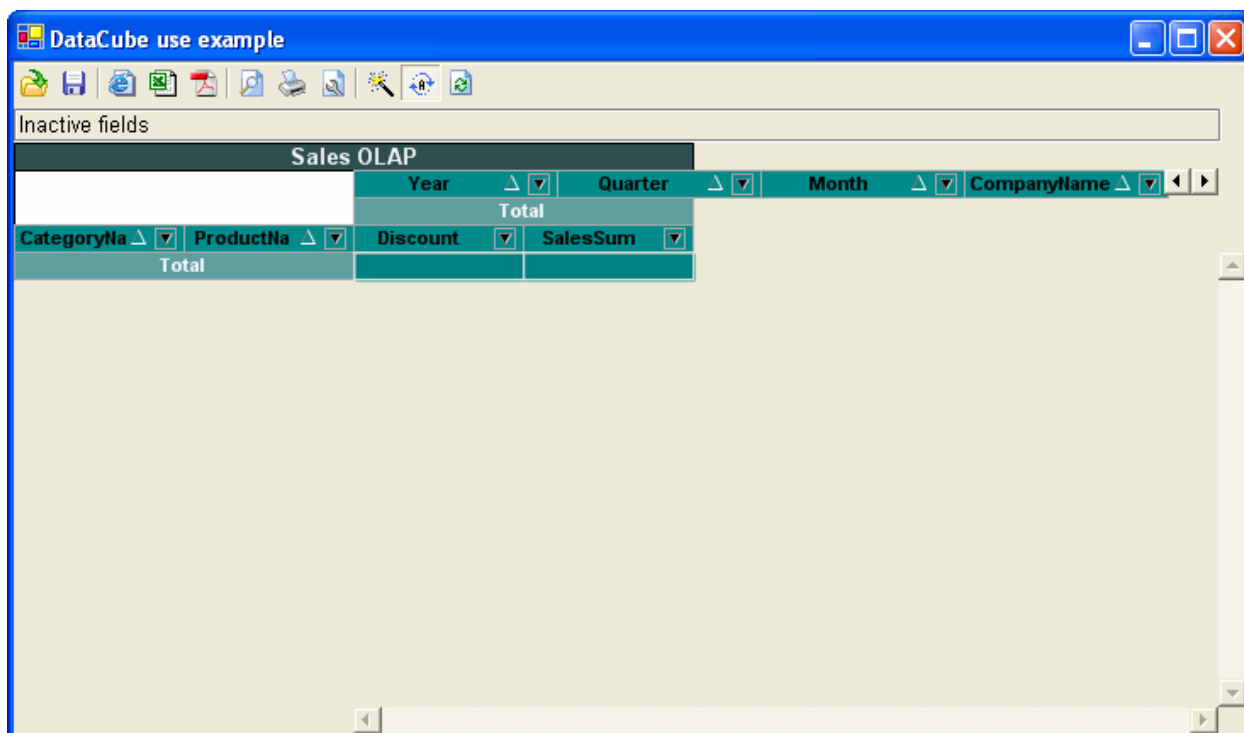
Let us create a Windows Forms application and add the DataCubeGrid component object situated on SharpShooter OLAP tab onto the form.



In order to work with DataCubeGrid, it is necessary to specify a DataCube to it. Let us create a DataSet and a DataCube by adding them onto the form. Adjust the DataCube and the DataSet as it is shown in the previous example of this User Guide.

Let us set the DataCube ShowSingleFact property to true. This will ensure the display of a fact header provided that the quantity of fact is equal to 1.

Let's specify the obtained DataCube as the DataCubeGrid Source property value.



It is necessary to add the code for filling the Sales table from the DataSet into the form load event.

It is also necessary to write the following code in the form load event handler:



salesDataCube.RaiseDataCubeChanged();

If you change DataCube settings from the code (not from DataCubeGrid), it is necessary to invoke the DataCube.RaiseDataCubeChanged() to notify DataCubeGrid about settings change. Only in this case the correct displaying of the changed pivot table will be provided.

Let us run our application. The required pivot table will be displayed on the form.

Inactive fields		Sales OLAP							
		Year	Month	Quarter	CompanyName				
		1996	June	2	Total				
		Total		Total		Total		Total	
CategoryName	ProductName	SalesSum	Discount	SalesSum	Discount	SalesSum	Discount	SalesSum	Discount
Meat/Poultry	Perth Pasties					5215.2	0.41%	21510.2	0.95%
	Pøti chinois	48	0.00%	48	0.00%	7584	0.74%	19512	2.11%
	Thüringer Rostbr					48278.1	1.36%	87736.4	1.69%
	Tourtierre					938.7	0.95%	5121	2.15%
	Total	48	0.00%	48	0.00%	74982	4.01%	178188.8	9.21%
Produce	Longlife Tofu					900	0.21%	2566	0.54%
	Manjimup Dried A					21730	0.78%	44742.6	1.65%
	Russle Sauerkrau	364.8	0.12%	364.8	0.12%	8436	0.25%	26865.6	1.03%
	Tofu	488.25	0.23%	488.25	0.23%	837	0.40%	8630.4	1.22%
	Uncle Bob's Orga	480	0.08%	480	0.08%	13560	0.33%	22464	0.66%
	Total	1333.05	0.44%	1333.05	0.44%	45463	1.97%	105268.6	5.11%
Seafood	Boston Crab Mea					5593.6	0.37%	19048.3	1.73%
	Carnarvon Tigers					11312.5	0.66%	31987.5	1.78%
	Escargots de Bou					2676.5	0.50%	6664.75	1.49%
	Gravad lax					1768	0.17%	3047.2	0.25%
	Ikura	31	0.00%	31	0.00%	11253	0.33%	22140.2	1.20%
	Inlagd Sill					5016	0.70%	14542.6	2.02%
	Jack's New Engla	501.8	0.00%	501.8	0.00%	3782.8	0.54%	9098.1	1.57%
	Konbu	84	0.04%	84	0.04%	4074	1.03%	5234.4	1.40%
	Nord-Ost Matjesh					6110.04	0.87%	14775.54	2.15%
	Rud Kaviar	30	0.01%	30	0.01%	180	0.01%	4200	0.55%
	Ruugede sild					1216	0.21%	4740.5	0.87%
Spegesild	396	0.14%	396	0.14%	1932	0.35%	6144	1.42%	
	Total	1042.8	0.19%	1042.8	0.19%	54914.44	5.73%	141623.09	16.42%
	Total	13995.05	2.43%	13995.05	2.43%	547247.6	37.38%	1354458.59	100.00%

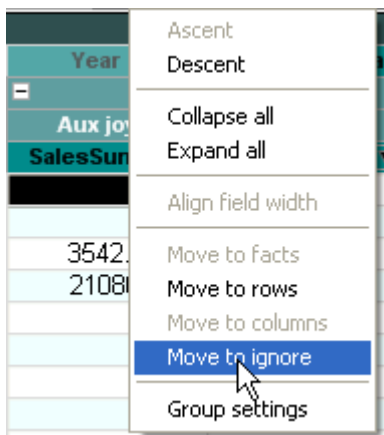
DataCubeGrid allows a user to implement visual control over a pivot table.

Let us save the pivot table settings to a file having specified the "Layout and sheet" file type in the saving dialogue box. Our task will be to form a pivot table to represent the sales total for a specific product, for a specific time period not taking any discounts into account. For a start, let us exclude the unnecessary fields from facts and dimensions. Here, these fields are Month, Quarter, CompanyName and Discount. We can exclude these fields from our pivot table calculation by dragging their headers into the DataCubeGrid Inactive fields scope.

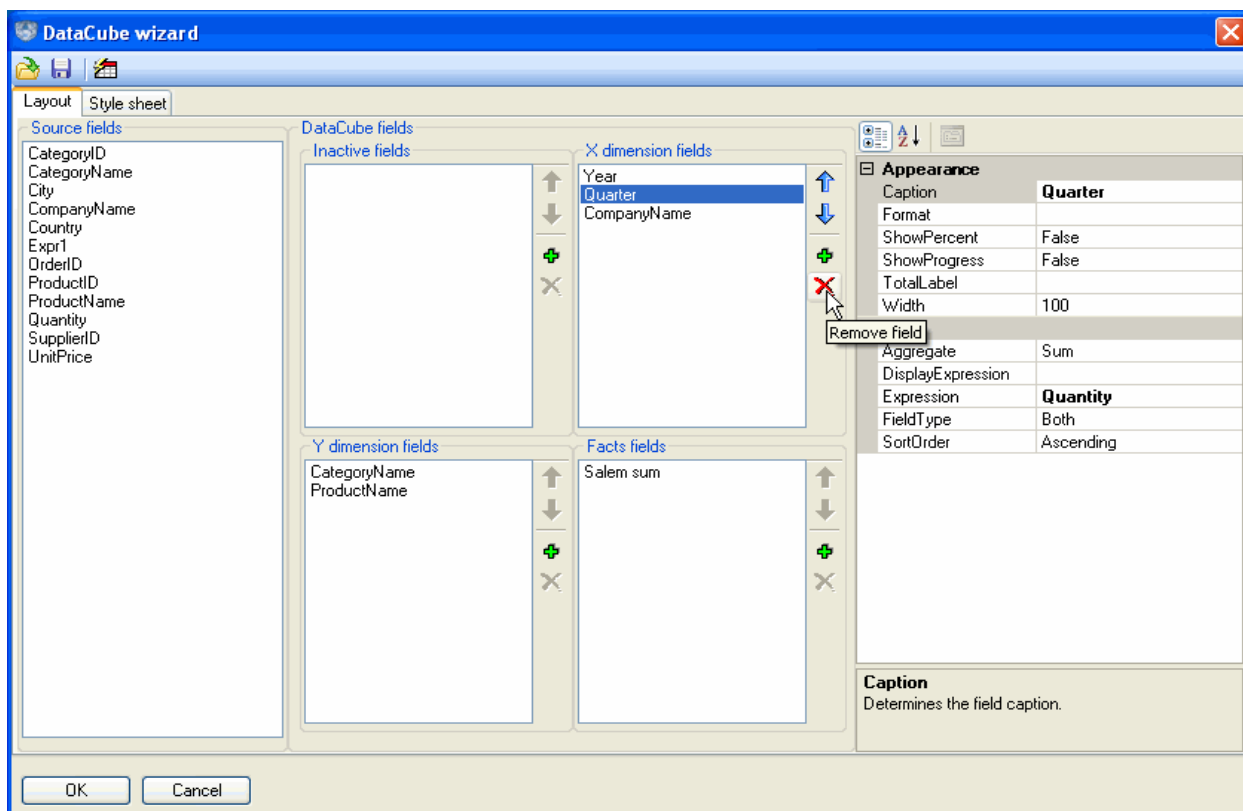


		Sales OLAP								
		Year		CompanyName						
		1994								
		Aux joyeux ecclésiastiques		Bigfoot Breweries		Cooperativa de Quesos 'Las Cabras'		Escargots Nouvea		
CategoryName	ProductName	SalesSum	Discount	SalesSum	Discount	SalesSum	Discount	SalesSum	Discount	
Beverages	Chai									
	Chang									
	Chartreuse verte	3542.4	0.29%							
	Côte de Blaye	21080	0.25%							
	Guaraná Fantástico									
	Ispoh Coffee									
	Lakkalikööri									
	Laughing Lumber				56	0.21%				
	Outback Lager									
	Rhinbrandt Klost									
Condiments	Sasquatch Ale			896	0.04%					
	Steeleye Stout			2937.6	0.12%					
	Total	24622.4	0.54%	3889.6	0.37%					
	Aniseed Syrup									
	Chef Anton's Caj									

Another way of doing that is to click on a field with the right mouse button and select the Move to ignore item from the contextual menu.

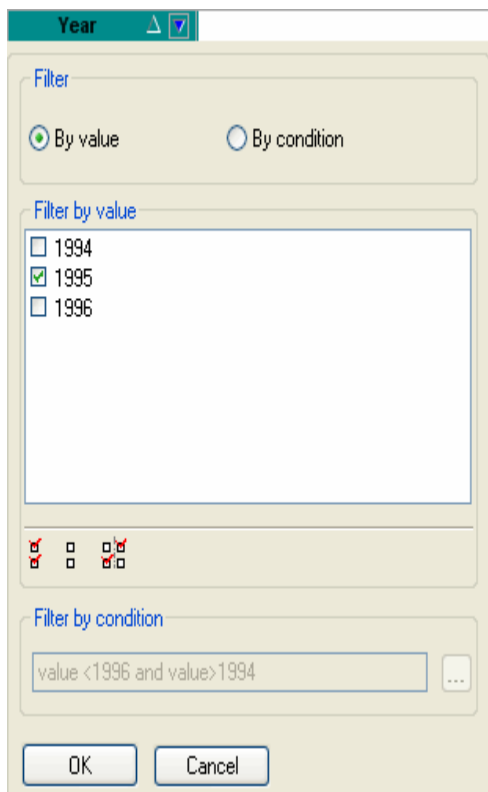


If the specified fields are no longer needed for calculations, they can be excluded from the table by means of launching the DataCube wizard designer and deleting these fields from the corresponding field lists.



Let us also modify the Expression property for the Sales sum field by assigning the "UnitPrice \* Quantity" value to it. This will expel the participation of the discount value in calculations. Close the designer by clicking "OK".

Let us use the filters capabilities to specify the pivot table formation cycle. Click the ( ) button on the Year field header to call the filter dialogue. Here, let us select filtering by condition. For example: "value < 1996 and value > 1994".





As a result, we shall get the required pivot table.

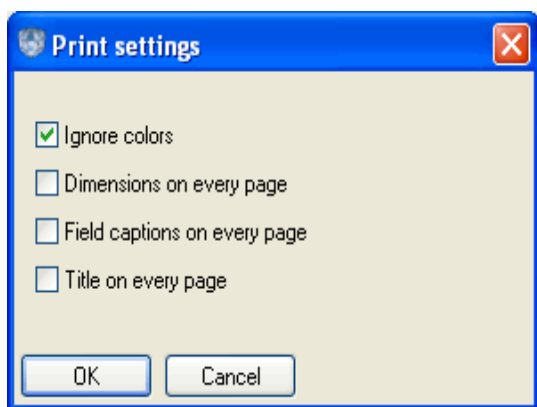
Sales OLAP			
		Year	Total
Beverages	Chai	1,99%	1,99%
	Chang	1,91%	1,91%
	Chartreuse verte	1,58%	1,58%
	Côte de Blaye	1,16%	1,16%
	Guaranã Fantôsti	1,74%	1,74%
	Iroh Coffee	0,25%	0,25%
	Lakkalikööri	1,66%	1,66%
	Laughing Lumber	0,00%	0,00%
	Outback Lager	1,99%	1,99%
	Ruuhikukka Koste	1,74%	1,74%
	Sasquatch Ale	1,16%	1,16%
	Steeleye Stout	2,07%	2,07%
	<b>Total</b>	<b>17,25%</b>	<b>17,25%</b>
Condiments	Aniseed Syrup	0,17%	0,17%
	Chef Anton's Caj	1,49%	1,49%
	Chef Anton's Gu	0,17%	0,17%
	Genen Shouyu	0,17%	0,17%
	Grandma's Boyse	0,00%	0,00%

Now let us open the pivot table settings previously saved. We can make any necessary changes of our pivot table appearance, i.e. roll the unnecessary groups up and change the sorting. We can do that by using the corresponding buttons or by selecting the required item from a field's contextual menu.

Sales OLAP									
		Year	Quarter	Month	CompanyName				
		1995							
		3							
		Сентябрь							
		Svensk Sjuåda AB		Tokyo Traders		Total		Total	
CategoryName	ProductName	SalesSum	Discount	SalesSum	Discount	SalesSum	Discount	SalesSum	Discount
Produce	Total					4455	0,21%	12336	
	Boston Crab Mea					1104	0,00%	2833,6	
	Carnarvon Tigers					5125	0,08%	8750	
	Escargots de Bou					79,5	0,00%	477	
	Gravad lax								
	Ikura			558	0,21%	558	0,21%	3193	
	Inlagd Sill	285	0,00%			285	0,00%	2204	
Seafood	Jack's New Engla							202,65	
	Konbu					78	0,12%	486	
	Nord-Ost Matjesh					776,7	0,00%	1294,5	
	Rud Kaviar	525	0,12%			525	0,12%	1020	
	Rugede sild					190	0,00%	190	
	Spegesild					276	0,29%	276	
<b>Total</b>		810	0,12%	558	0,21%	8997,2	0,83%	20926,75	
<b>Total</b>		810	0,12%	558	0,21%	49981,69	4,01%	144534,62	

This table with its current settings can be printed and/or exported into Excel table, PDF or HTML document.

The table printing occurs subject to current print settings. Let's change these settings by calling the Print settings dialogue.



We shall change the settings in the following way:

We shall set the IgnoreColor (the printing occurs ignoring the preset pivot table presentation style) and Field captions on every page (the pivot table fields captions will appear on every page of a document).

Also we shall cancel Dimensions on every page (the dimension groups will be printed only once) and Title on every page (the pivot table header will be printed only once).

After we do that, the print preview and the printing itself will be performed subject to these settings.

This example can be found in the DataCubeGridUseExample catalog.

### Working with Groups

The capability of data grouping in pivot table dimensions allows getting more detailed reporting. The SharpShooter OLAP enables to group data in dimensions by default taking into consideration sequence of assigning the fields representing these data. In order to provide extra flexibility the SharpShooter OLAP grants the capability to set customs rules for groupings. Let's consider the example demonstrating this functionality. Let us assume that we need to get the detailed information on purchases by supplying countries and part of the world these countries belong to.

Let's create a WindowsForms application and add the DataCube component's object onto the form. Let us name a newly-created object salesDataCube. Let's adjust the salesDataCube object so that a pivot table layout looks like it is shown below:

Sales					
	CategoryName	ProductName			
	CategoryName1				Total
Country	ProductName1	ProductName2	ProductName3	Total	
Country1					
Country2					
Total					

As a result we should get the following pivot table:



City	Quantity	Sum	Sales			
	CategoryName	ProductName	Beverages			
Country	Chai	Chang	Chartreuse vert	Côte de Blaye	Guaraná Fantá	Ipoh Coffee
Australia						
Brazil					216,00	
Canada						
Denmark						
Finland						
France			500,40	5902,40		
Germany						
Italy						
Japan						
Netherlands						
Norway						
Singapore						1205,20
Spain						
Sweden						
UK	651,60	786,60				
USA						
Total	651,60	786,60	500,40	5902,40	216,00	1205,20

To specify the required grouping let's create one more field for a pivot table with the help of DataCube wizard. Let us assign the following properties for a newly-created field:

Caption = "Part of world"

Expression = "Country"

Appearance	
Caption	Part of world
Format	
ShowPercent	False
ShowProgress	False
TotalLabel	
Width	120
Data	
Aggregate	Sum
DisplayExpression	
Expression	Country
FieldType	Both
SortOrder	Ascending

We get the following pivot table



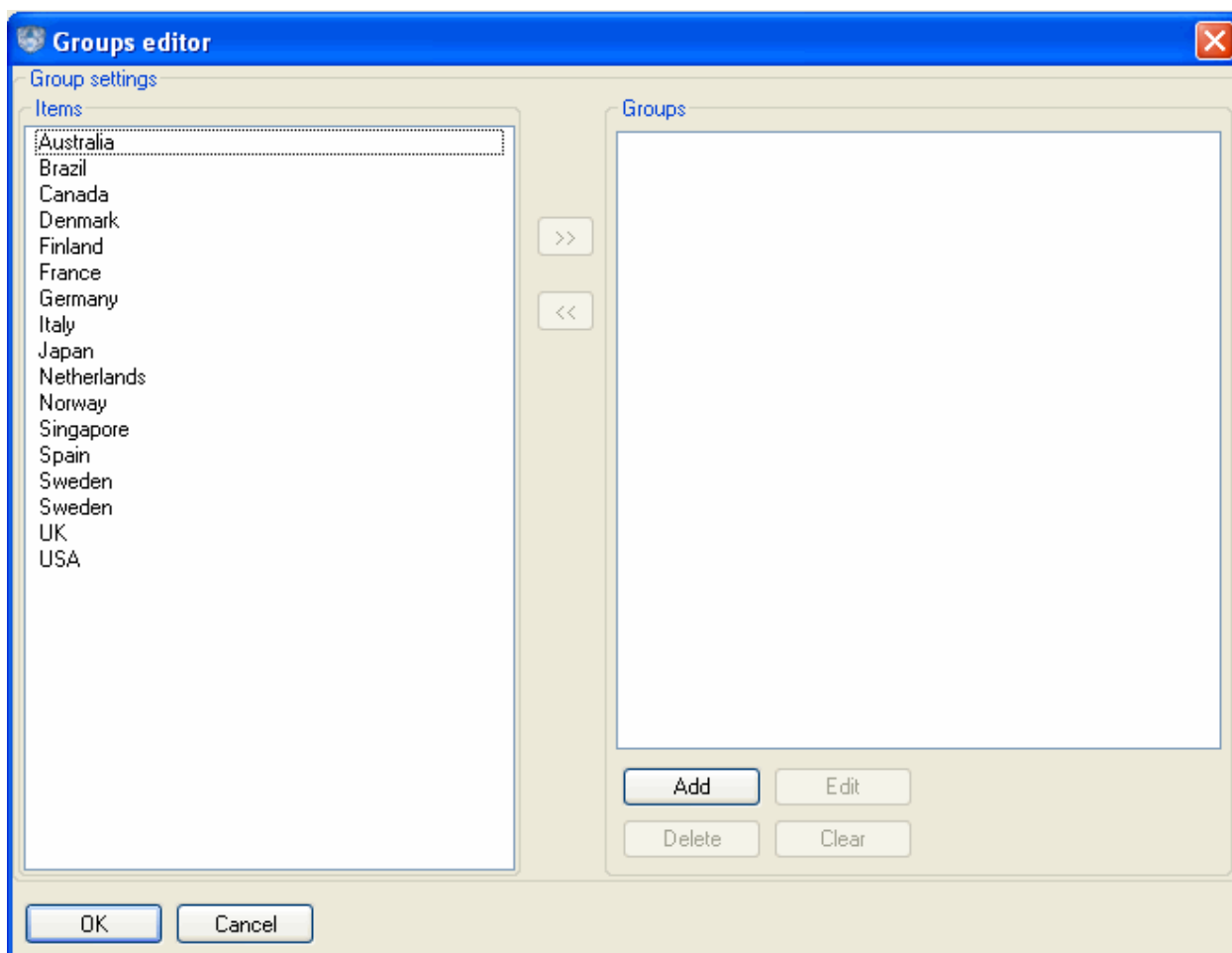


City		Quantity	Sum		
Sales					
		CategoryName	ProductName		
		Seafood			Total
Part of world	Country	Rugede sild	Spegesild	Total	Total
Australia	Australia			1 612,50	7 984,00
	Total			1 612,50	7 984,00
Brazil	Brazil				216,00
	Total				216,00
Canada	Canada				3 878,85
	Total				3 878,85
Denmark	Denmark	129,20	300,00	429,20	429,20
	Total	129,20	300,00	429,20	429,20
Finland	Finland				1 199,90
	Total				1 199,90
France	France			227,90	11 030,50
	Total			227,90	11 030,50
Germany	Germany			776,58	9 133,11
	Total			776,58	9 133,11
Italy	Italy				4 461,30

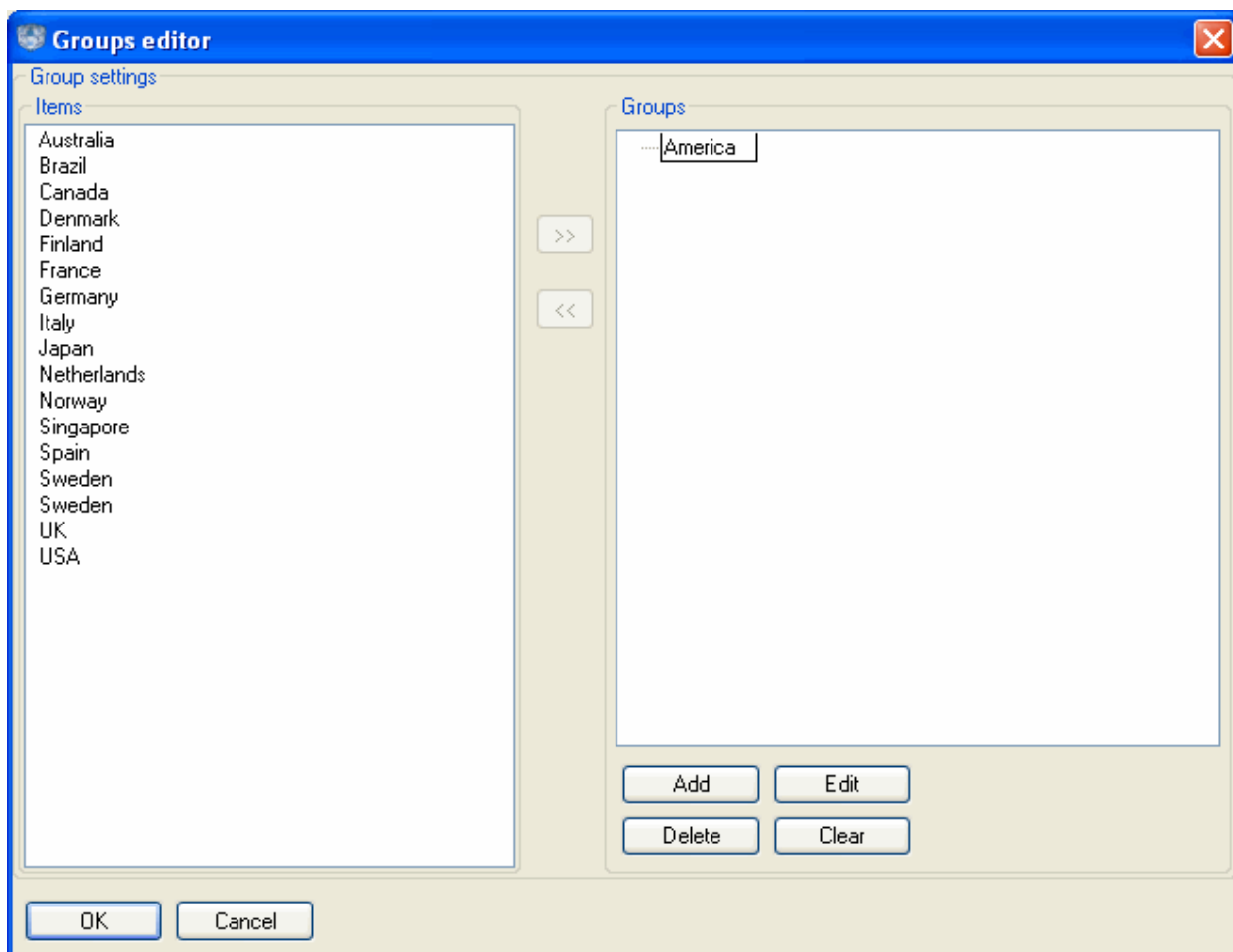
Now let's select the Group Settings item in the contextual menu of the Part of World field.

City		Quantity	Sum		
Sales					
		CategoryName	ProductName		
		Seafood			Total
Part of world	Country	Rugede sild	Spegesild	Total	Total
Australia	Australia			1 612,50	7 984,00
	Total			1 612,50	7 984,00
Brazil	Brazil				216,00
	Total				216,00
Canada	Canada				3 878,85
	Total				3 878,85
Denmark	Denmark	129,20	300,00	429,20	429,20
	Total	129,20	300,00	429,20	429,20
Finland	Finland				1 199,90
	Total				1 199,90
France	France			227,90	11 030,50
	Total			227,90	11 030,50
Germany	Germany			776,58	9 133,11
	Total			776,58	9 133,11
Italy	Italy				4 461,30

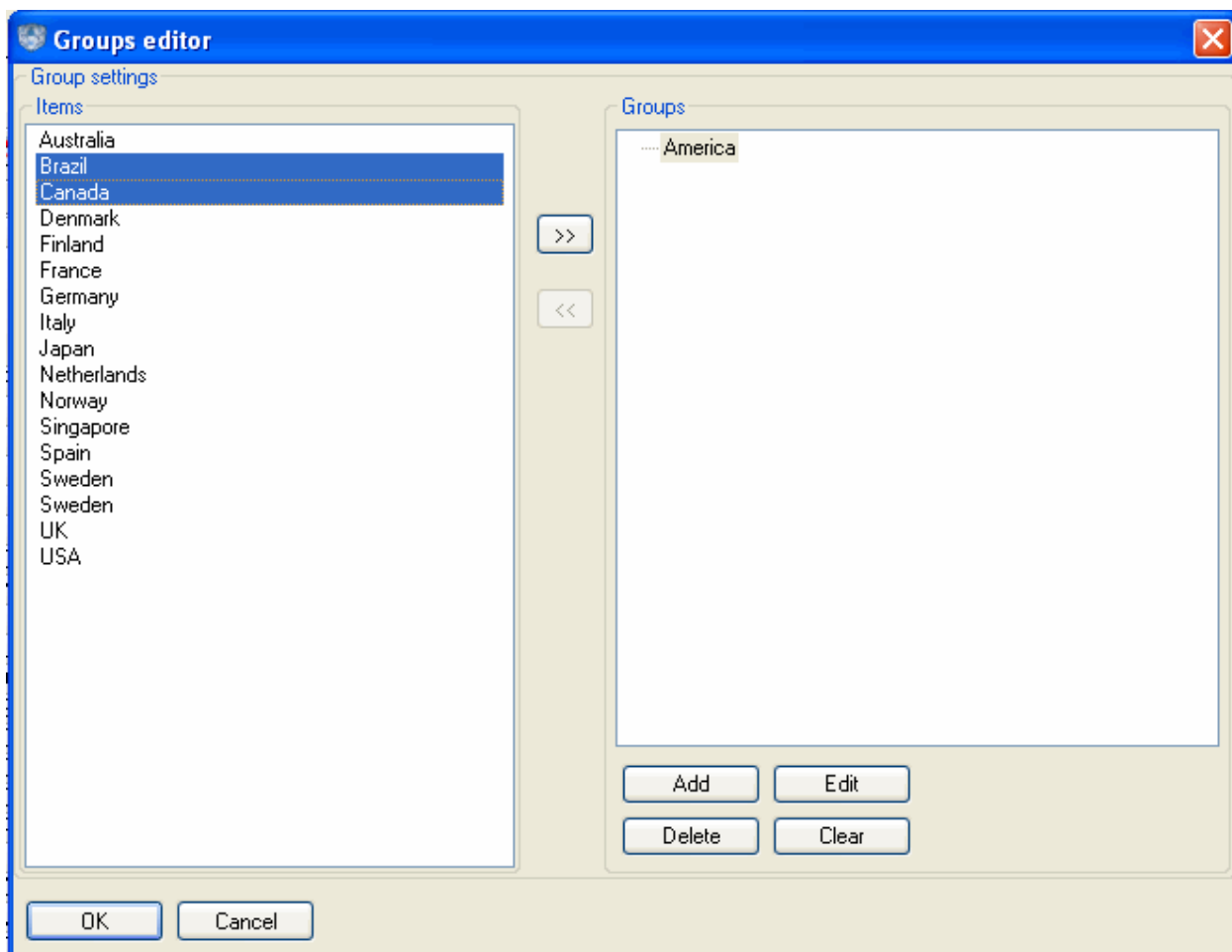
This action opens the group settings dialogue box for this field.



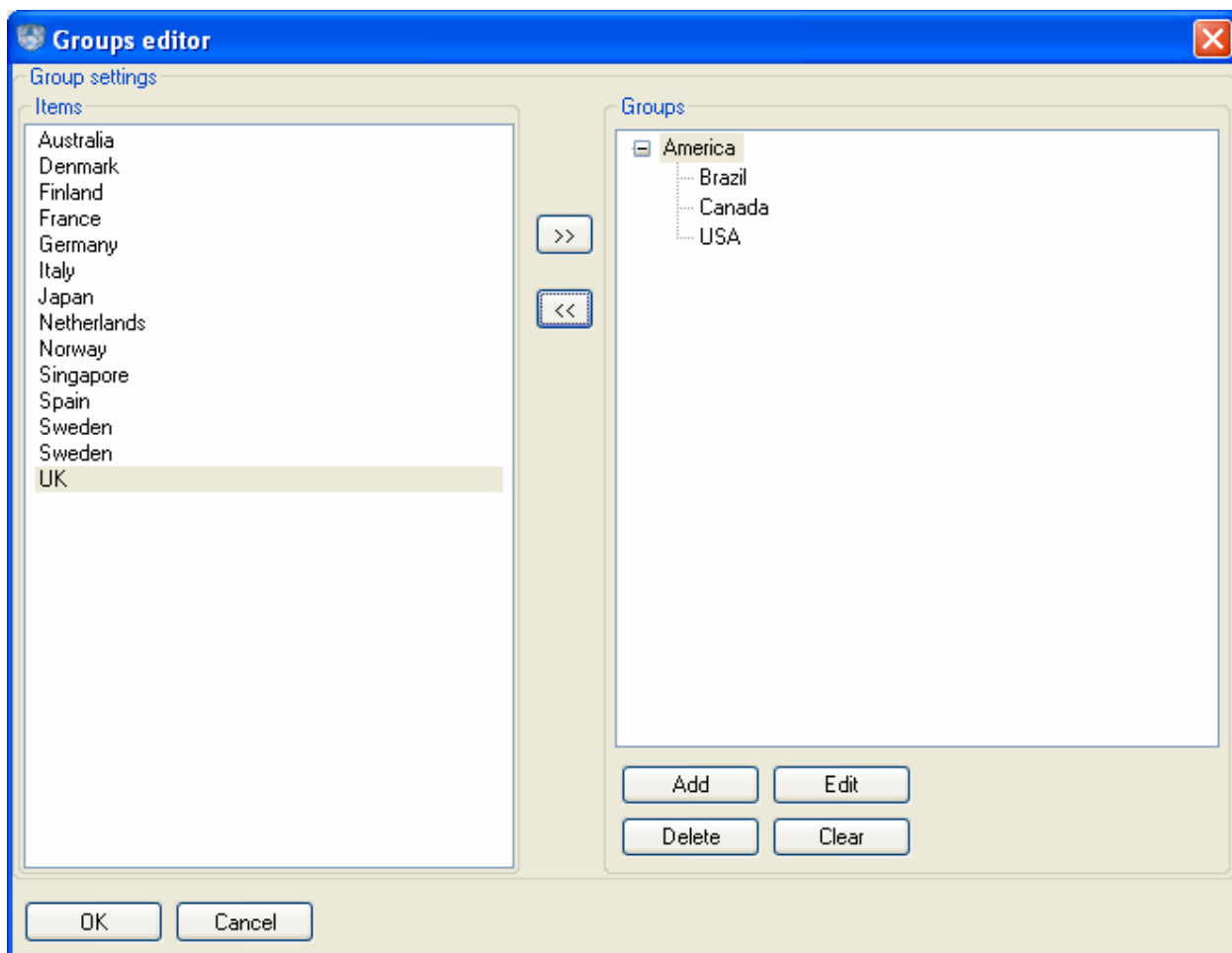
In the Items list all field values obtained as a result of its expression calculation are shown. In this particular case they are the countries. In order to group these countries by parts of the world it is necessary to add the corresponding groups. Let's press the "Add" button and specify group's name, for example America.



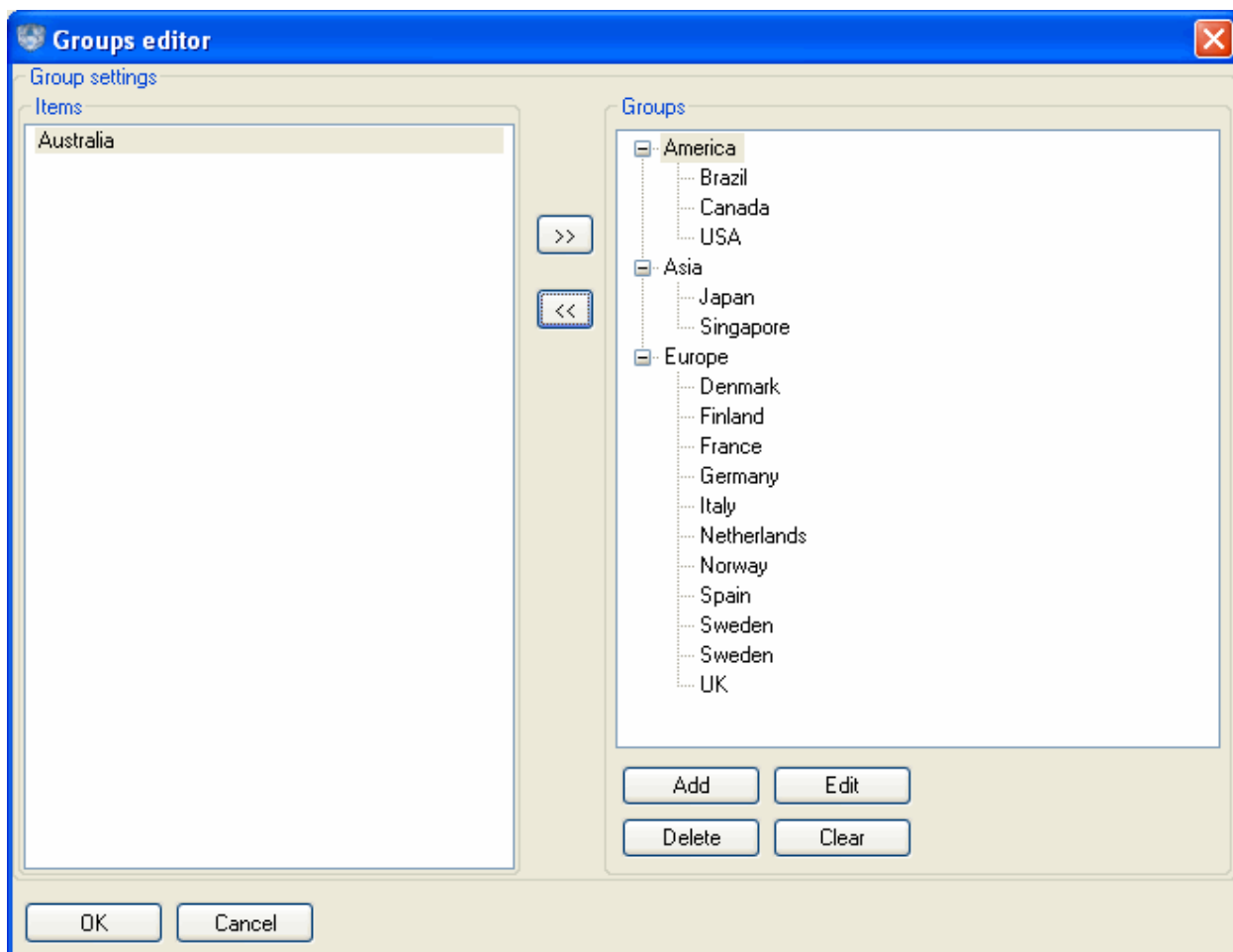
Let's add Brazil, Canada and USA into the group. To do that we should select the required countries and drag them by means of mouse to the required group as it is shown on the picture below.



After the values are added to the group they are eliminated from the list and displayed in the groups' tree like the leaves.



Let's add Asia and Europe in the same way.



Press the button "OK" to confirm the settings. As a result the pivot table will look like it is shown below:

City		Quantity	Sum		Sales	
		CategoryName	ProductName		Seafood	
Part of world	Country	Röd Kaviar	Røgede sild	Spegesild	Total	Total
America	Brazil					216,00
	Canada					3878,85
	USA				1138,40	5128,70
	Total				1138,40	9223,55
Asia	Japan				1210,00	2346,05
	Singapore					2036,60
	Total				1210,00	4382,65
Australia	Australia				1612,50	7984,00
	Total				1612,50	7984,00
Europe	Denmark		129,20	300,00	429,20	429,20
	Finland					1199,90
	France				227,90	11030,50
	Germany				776,58	9133,11
	Italy					4461,30
	Netherlands					263,30
	Norway					1941,20
	Spain					1261,60
	Sweden	201,00			896,20	1349,20
	UK					3841,40
Total		201,00	129,20	300,00	2329,88	34910,71
Total		201,00	129,20	300,00	6290,78	56500,91

You can find this example in the CustomGroupingExample folder.



Extra flexibility in creating group can be obtained by means of expressions.

For instance, the countries grouping shown in the previous example can be carried out by the first letters of the countries names. To do that it is necessary to create Product Group field and specify "subString(ProductName, 0, 1)" value to the Expression property. The result of such grouping is shown below.

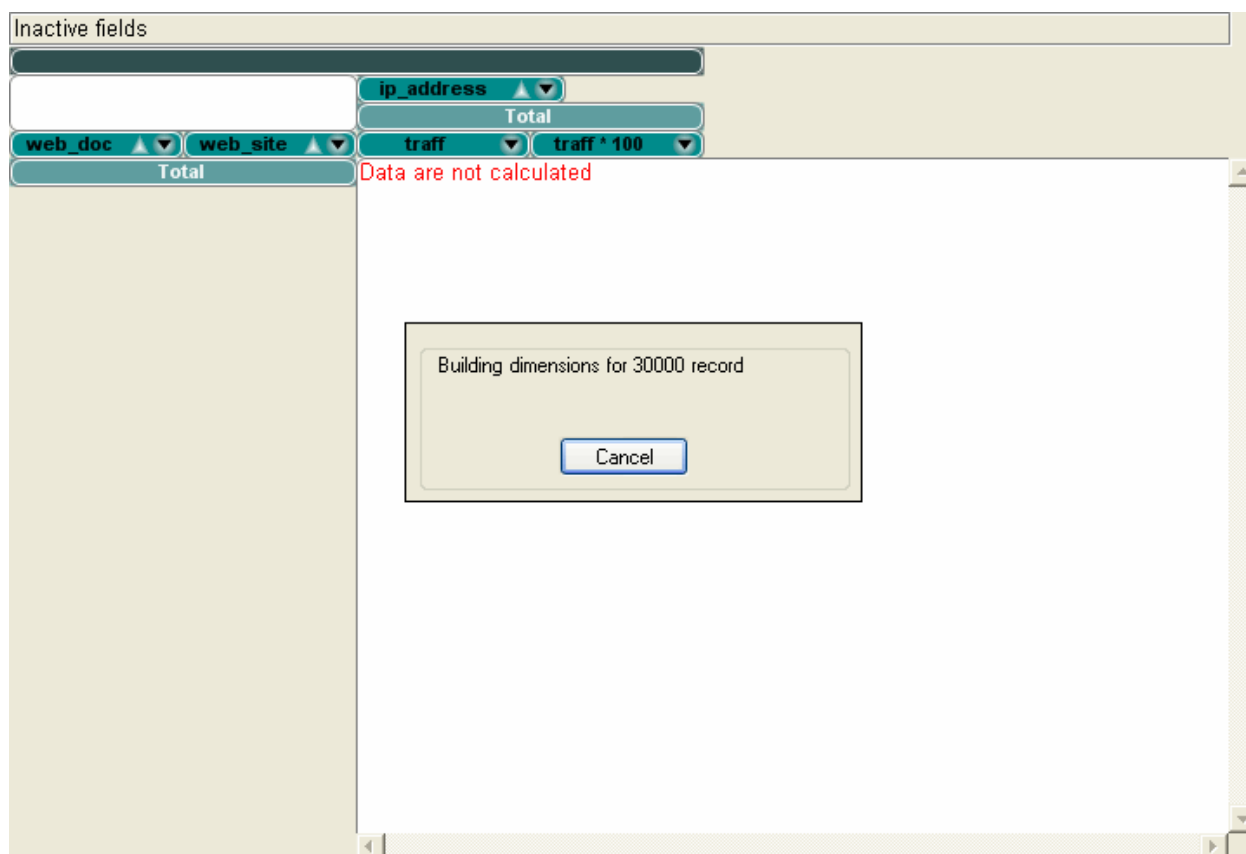
City		Quantity	Sum				
Sales							
		CategoryNam	ProductName				
		Beverages					
Country grou	Country	Chai	Chang	Chartreuse vert	Côte de Blaye	Guaraná Fantá	
+	A	Total					
+	B	Total				216,00	
+	C	Total					
+	D	Total					
+	F	Total		500,40	5902,40		
=	N	Netherlands					
		Norway					
		Total					
=	S	Singapore					
		Spain					
		Sweden					
		Total					
=	U	UK	651,60	786,60			
		USA					
		Total	651,60	786,60			
Total			651,60	786,60	500,40	5902,40	216,00

To specify groups it is also possible to use the Range function defined in the expression language (see *The Description of Expression Language Syntax* section).



### The Work with Large Body of Data

SharpShooter OLAP allows creating pivot tables using large body of data. The SharpShooter OLAP is supplied with a number of specific instruments to make the product use in such cases more convenient.

Pivot table calculation can be carried out in a standalone thread. It allows a user to interrupt calculation at any time. While working with DataCubeGrid it is carried out the following way: when pivot table recalculation is started there appears a dialogue box displaying calculation progress. The dialog box also has the "Cancel" button by pressing on which a user can correctly interrupt the calculation.



In order to on/off simultaneous calculation of the pivot table, use the `DataCubeGrid.CalculationMode` property.

DataCubeGrid interactivity can be temporarily turned off. So, when you specify any setting that requires table recalculation, calculation is not executed. As a result, while working with large body of data a user can adjust pivot table layout, set the required filters, group the values and only after that call forced recalculation. To make this operation available the Recalculate (  ) and Auto recalculate (  ) buttons are added on the DataCubeGrid Toolbar. The last button is responsible for turning off/on interactivity.

### Custom Cell Styles

When using the DataCubeGrid component, there is the possibility to selectively change the cell style set in the style sheet for the displayed DataCube. In order to change the cell style, one should use the DrawCell event of the DataCubeGrid object.

The DrawCell event is called each time a DataCubeGrid element is drawn. At that, the event handler receives the DrawCellEventArgs object as an argument.

Use the following DrawCellEventArgs object properties:

**CellStyle** – the PerpetuumSoft.Olap.CellStyle type object which contains the information on the current style of a table element drawn. Change this property to specify a custom cell display style;

**CellInfo** –the information on a table cell. You can learn more about the PerpetuumSoft.OlapCellInfo type in the Class reference;

**IsActiveCell** –indicates whether the cell specified is an active table cell.

A sample of the DrawCell event handler is given below.

```
private PerpetuumSoft.Olap.CellStyle customStyle = new CellStyle(Color.Purple,
Color.Black);
```





```
private void dataCubeGrid1_DrawCell(object sender,
PerpetuumSoft.Olap.DrawCellEventArgs e)
{
    if (e.CellInfo.XFieldItem != null && e.CellInfo.YFieldItem != null)
    {
        bool contains = CanChangeStyle(e.CellInfo.XFieldItem, textBox1.Text);
        if (contains && !e.IsActiveCell)
            e.CellStyle = customStyle;
        contains = CanChangeStyle(e.CellInfo.YFieldItem, textBox2.Text);
        if (contains && !e.IsActiveCell)
            e.CellStyle = customStyle;
    }
}

private bool CanChangeStyle (PerpetuumSoft.Olap.FieldItem item, string text)
{
    PerpetuumSoft.Olap.FieldItem fi = item;
    while(fi != null && fi.Value != null)
    {
        if ((text.Length > 0) && (fi.Value.ToString().IndexOf(text) >= 0))
            return true;
        fi = fi.Parent;
    }
    return false;
}
```

You can find a sample of the program which implements a custom cell style in the DataCubeGridCustomDrawing catalogue.

### Custom Drawing

Along with the ability to change styles of table elements while using the DataCubeGrid component, a user can draw either one or another table element with his/her own hands. At an attempt to draw a grid cell, the DataCubeGrid.DrawGridItem event is called. So, in order to change a specific table cell drawing method, one should implement a handler for this event. The DrawGridItemEventArgs object is passed to the event handler as an argument.

Use the following DrawGridItemEventArgs object properties:

Graphics – the System.Drawing.Graphics type object actual for table drawing;

Bounds – the System.Drawing.Rectangle type object which contains the information on the position of an element within a table;

CellStyle – the PerpetuumSoft.Olap.CellStyle, type object which contains the information on the current style of a table element drawn;

CellInfo – the information on a table cell. You can learn more about the PerpetuumSoft.Olap.CellInfo type in the Class reference;

Handled – if this property is set to true, the DataCubeGrid will not draw an element itself.

A sample of the DrawGridItemCell event handler is given below.

```
private void dataCubeGrid1_DrawGridItem(object sender,
PerpetuumSoft.Olap.DrawGridItemEventArgs e)
{
    if ((e.CellInfo.XFieldItem != null && e.CellInfo.YFieldItem == null &&
e.CellInfo.XFieldItem.Value.ToString().IndexOf(textBox1.Text) >= 0 &&
textBox1.Text.Length > 0)
        || (e.CellInfo.XFieldItem == null && e.CellInfo.YFieldItem != null &&
e.CellInfo.YFieldItem.Value.ToString().IndexOf(textBox2.Text) >= 0 &&
textBox2.Text.Length > 0))
```



```
{
    Brush b = new System.Drawing.Drawing2D.LinearGradientBrush(e.Bounds,
Color.White, Color.Purple, 30);
    e.Graphics.FillRectangle(b, e.Bounds);
    e.Graphics.DrawString(e.Value, new Font("Arial", 8, FontStyle.Bold),
Brushes.Black, e.Bounds);
    e.Handled = true;
}
}
```

You can find a sample of the program which implements the use of custom drawing in the DataCubeGridCustomDrawing folder.

The capabilities supplied by the aforementioned events (i.e. DataCubeGrid.DrawCell and DataCubeGrid.DrawGridItem) are enough to provide your table with custom-made appearance suitable for a particular task. However, in case these instruments are not enough, you can assign a completely custom drawing to the DataCubeGrid object. To do that, you need to create a class derived from the abstract PerpetuumSoft.Olap.Painter class and specify it as the primary pivot table painter (myDataCubeGrid.Painter = myCustomPainter).

### The Use and Capabilities of DataCubePrintDocument

The DataCubePrintDocument component is responsible for pivot table output onto a printing device.

Let us create a Windows Forms application and add the DataCubePrintDocument component object situated on the SharpShooter OLAP tab onto the form.

It is necessary to specify a DataCube to the DataCubePrintDocument in order to work with it latter. Let us create a DataSet and a DataCube by adding them onto the form and adjust them as it is shown in the "The Use and Capabilities of DataCube" section of this User Guide.

Let's specify the obtained DataCube as the DataCubePrintDocument DataCube property value.

We shall add the following code into the DataCubePrintDocument object BeginPrint event:

```
MessageBox.Show("Start of printing.");
```

And the following for BeginPrint:

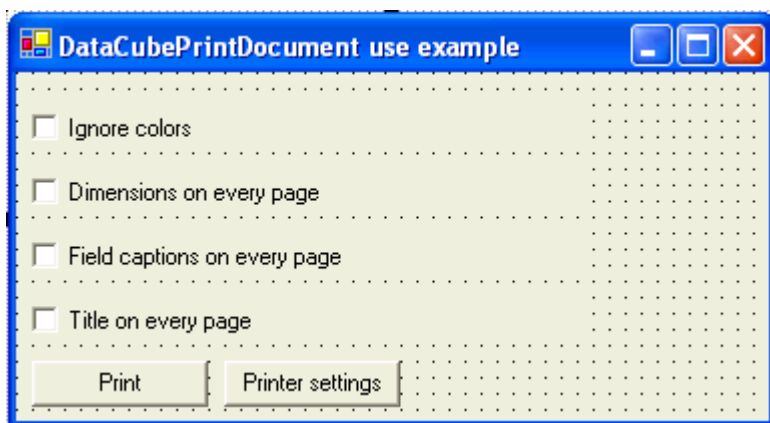
```
MessageBox.Show("End of printing.");
```

It is necessary to add the code for filling the Sales table from the DataSet into the form filling event.

We shall also add the following four System.Windows.Forms.CheckBox elements onto the form

```
isMultiColorCheckBox;
dimensionsOnEveryPageCheckBox;
titleOnEveryPageCheckBox;
fieldCaptionsOnEveryPageCheckBox;
```

And specify their properties as it is shown in the image below.



Now let us add the `System.Windows.Forms.PrintDialog` component onto the form and set the `DataCubePrintDocument` object as the `Document` property value.

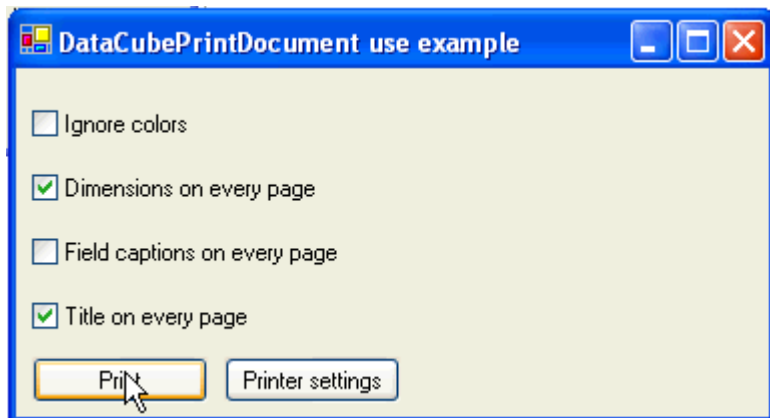
Add the `System.Windows.Forms.Button` component `printButton` object onto the form. Then write the following code in the button press event handler:

```
PerpetuumSoft.Olap.DataCubePrintSettings settings = new
PerpetuumSoft.Olap.DataCubePrintSettings();
settings.IgnoreColors = isMultiColorCheckBox.Checked;
settings.DimensionsOnEveryPage = dimensionsOnEveryPageCheckBox.Checked;
settings.TitleOnEveryPage = titleOnEveryPageCheckBox.Checked;
settings.FieldCaptionsOnEveryPage = fieldCaptionsOnEveryPageCheckBox.Checked;
dataCubePrintDocument.PrintSettings = settings;
dataCubePrintDocument.PrinterSettings = printDialog.PrinterSettings;
dataCubePrintDocument.Print();
```

Add the `System.Windows.Forms.Button` component `printerSettingsButton` object onto the form and write the following code in the button press event handler:

```
printDialog.ShowDialog();
```

Run the application.



Now let us specify all necessary pivot table print settings. Press the "Printer settings" button to specify the required printer settings. After that, click the "Print" button. The pivot table will be printed.

This example can be found in the `DataCubePrintDocumentUseExample` folder.

### The Interaction of the SharpShooter OLAP with Third Party Components

The SharpShooter OLAP is intended for creating and utilizing cross reports based on multiple statistical data of large volumes. The SharpShooter OLAP provides an opportunity to represent the final report data not only as a table; it also allows organizing the information so to make it



available for use by third party chart components. Presently, there are lots of such components available. These are various components for charts, diagrams, reports and grids creation.

The DataCubeView component which is included in the SharpShooter OLAP package can be used for implementing an interface similar for a pivot table and graphic visualization components. DataCubeView allows creating presentations of data stored within the DataCube object. The DataCubeView component "expands" a pivot table dimension cube presenting the data as a linear table the columns of which are facts of the initial cube and a field of one of dimensions. DataCubeView makes it possible to move from one dimension into another and from one group into another within one and the same dimension.

It is more preferable to use the DataCubeView component in order to organize the presentation of pivot table data to other components. However, there are many graphic visualization components that do not correctly implement the interfaces necessary for using presentations as data sources. Due to that fact, SharpShooter OLAP supplies a tool for interaction with other components with the help of ADO.NET objects.

The DataCubeView component is able to partially rollout a pivot table presented by it into ADO.NET objects which an end user can use at his own discretion.

### *The Interaction with SharpShooter OLAP by Means of ADO.NET Objects*

In the DataCube object a user has access to the Fill() open method. The Fill() open method implements pivot table data rollout into ADO.NET data objects.

The Fill() method is overloaded. Let us scrutinize each way of its use.

`PerpetuumSoft.Olap.DataCube.Fill(PerpetuumSoft.Olap.ViewDimension primaryDimension, System.Data.DataSet dataSet, int depth)` executes pivot table data rollout by the dimension that is defined by the primaryDimension list value (X - by columns, Y - by rows). Data rollout is conducted in the dataSet object subject to the max group nesting depth within a dimension (the depth parameter). If necessary, here takes place the creation of tables and adding these tables into the dataset. The name of each table corresponds to the name of a field responsible for group value calculation. The depth parameter value should not exceed the number of groups in a specified dimension.

`PerpetuumSoft.Olap.DataCube.Fill(PerpetuumSoft.Olap.ViewDimension primaryDimension, System.Data.DataSet dataSet, params string[] tableNames)` executes pivot table data rollout into the dataSet by the dimension that is defined by the primaryDimension list value (X - by columns, Y - by rows). At that, the tables assigned by the string name array are created in the dataset. The number of table names in the array shall not exceed the number of groups in a specified dimension.

`PerpetuumSoft.Olap.DataCube.Fill(PerpetuumSoft.Olap.ViewDimension primaryDimension, System.Data.DataSet dataSet, params System.Data.DataTable[] tables)` executes pivot table data rollout into DataTable objects contained in the tables array by the dimension that is defined by the primaryDimension list value (X - by columns, Y - by rows). The necessary requirement is the belonging of all tables to one DataSet object. The number of tables in the array shall not exceed the number of groups in a specified dimension.

A set of bound and filled tables represents the result of this or that Fill() method implementation result. The tables contain:

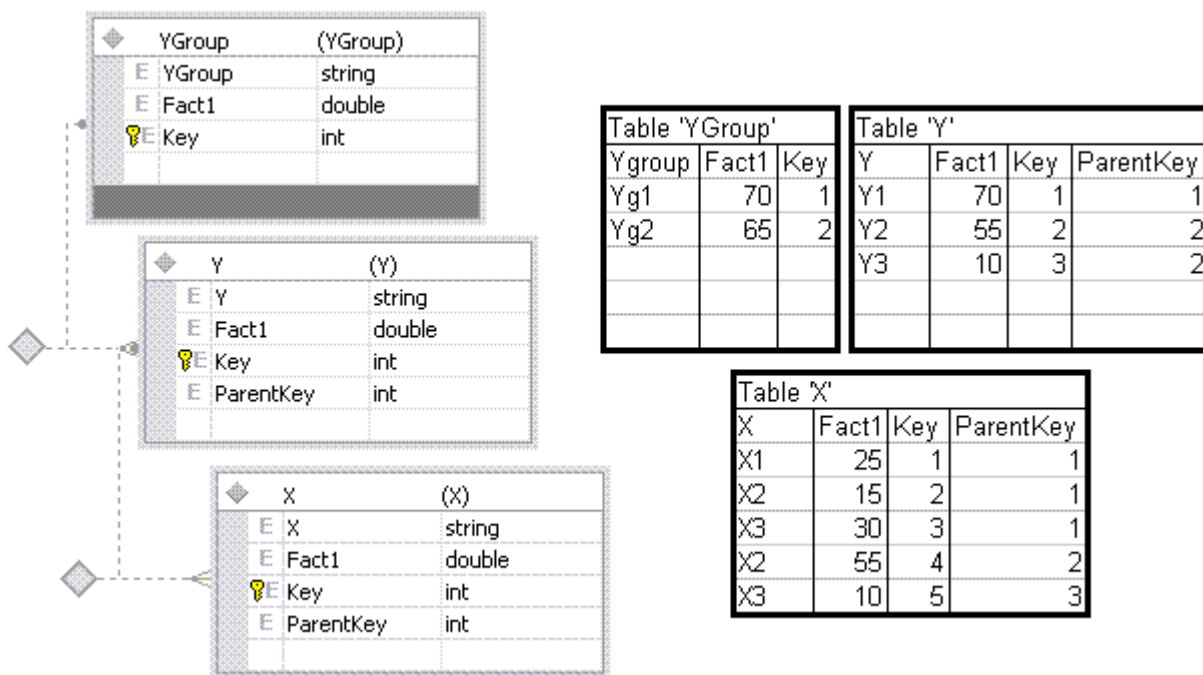
- group values in a specified dimension (a table column name corresponds to a name of a field responsible for group value calculation);
- fact values (a column name corresponds to a name of a field responsible for fact value calculation);
- primary and foreign key values which are used for organizing bonds between various tables.

At this method realization, the tables are being bound by the “one-to-many” type relation in such a way that the parent table corresponds to the dimension group with the nesting level lower than a group corresponding to a subordinate table.

Let us consider the pivot table shown in the image below.

			XGroup		
YGroup	Y	X	Xg1	Xg2	Total
Yg1	Y1	X1	25		25
		X2		15	15
		X3		30	30
		Total	25	45	70
Total			25	45	70
Yg2	Y1	X2		55	55
		Total		55	55
	Y3	X3		10	10
		Total		10	10
Total				65	65
Total			25	110	135

The `Fill(PerpetuumSoft.Olap.ViewDimension.Y, System.DataSet dataSet, 3)` method call (the `dataSet` does not contain tables and bonds) will lead to filling the `dataSet` object as follows:



### The Interaction with SharpShooter OLAP by Means of the DataCubeView Component

The `DataCubeView` object creates a presentation of a table specified to it as the `DataCube` property value. The `PrimaryDimension` property defining a value by which the cube “straightening” takes place is specified to the `DataCubeView` object.

While using the `DataCubeView` component a user has the ability to jump from one dimension group (parent) to nested subgroups (children).

Attention! In order to ensure the correct work of the `DataCubeView` object make sure that your components correctly support the following interfaces: `IList`, `IBindingList`, `ICollection`, `IEnumerable`, `ITypeedList`.

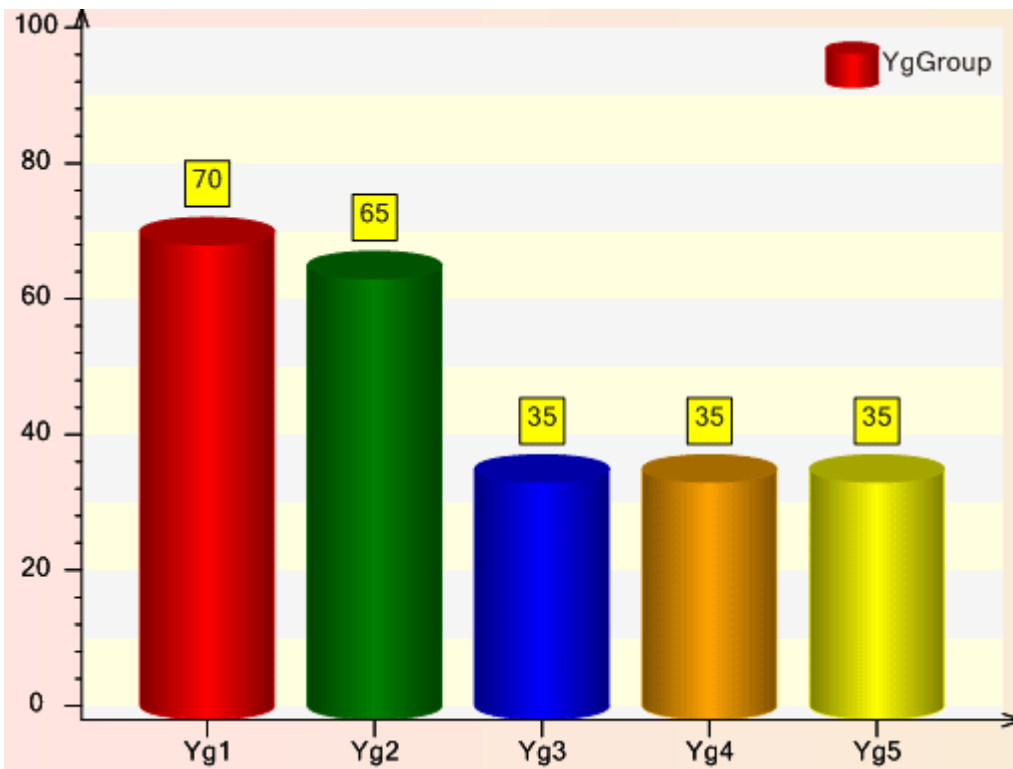
Example:

Let us add the DataCubeView object onto the form. As the DataCube property value we shall specify a pivot table. Let us examine the pivot table presented in the image below.

			XGroup		
YGroup	Y	X	Xg1	Xg2	Total
Yg1	Y1	X1	25		25
		X2		15	15
		X3		30	30
		Total	25	45	70
Total			25	45	70
Yg2	Y1	X2		55	55
		Total		55	55
	Y3	X3		10	10
		Total		10	10
Total				65	65
Total			25	110	135

Then we shall specify the PrimaryDimension property to the DataCubeView object. We shall specify Y. Now we can use the DataCubeView object for pivot table data presentation.

The result presented below is the output of Perpetuum Software SharpShooter Charts component which has used DataCubeView of a required pivot table as a data source.





## SharpShooter Charts

### Destination and Basic Features

SharpShooter Charts is intended for designing charts and diagrams of any type. The product includes the ChartViewer component designed for displaying charts in Windows Form applications.

Representing data in visual graphic form significantly streamlines data perception, but sometimes it is a very sophisticated task for a developer.

SharpShooter Charts allows developers to create charts of various types, customize their appearance, save built charts to various graphic formats and print them. This library provides stunning flexibility in work with data sources and offers the ability to create unbound charts.

Full-featured designer allows adjusting all chart constituents visually without the need to code. Created charts can be saved to a file for reuse.

### General Principles of the SharpShooter Charts

The main idea of SharpShooter Charts is to provide developers with the ability to create charts according to user requirements in a moment without challenging any special knowledge and high qualification. The product includes components allowing displaying and managing charts.

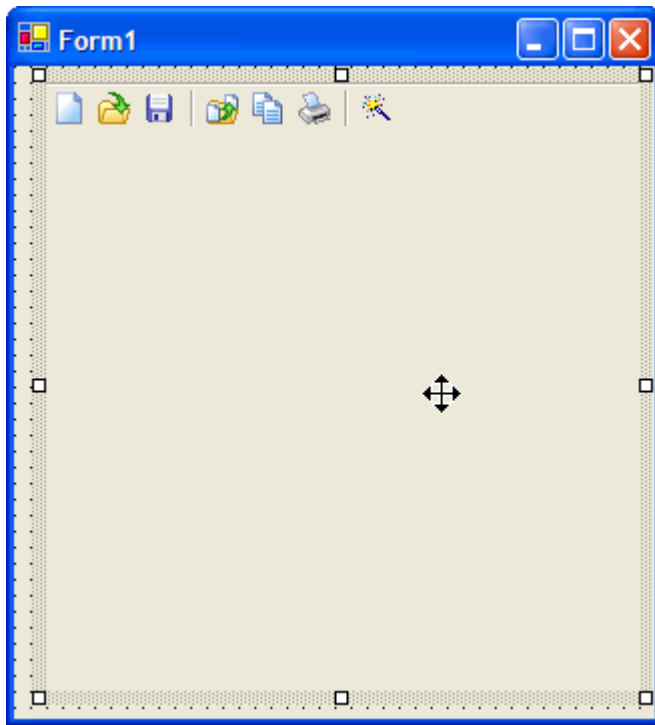
Chart is a compound object with a hierarchical structure. Primitives intended for assigning chart appearance and behavior are included in the chart.

Designer is used for charts creation. It allows charts design using simple elements, included in a chart in various combinations, and setting separate elements' properties with the help of convenient graphic means. In order to bind primitives and to assign more complicated functionality, expressions can be used. The ability to combine primitives in various combinations, flexibility in customizing elements' properties, and use of expressions all together allow developers to create charts according to customer requirements.

### Creation of a Simple Chart with the Use of Wizard

Let us create a simple chart with the SharpShooter Charts.

In order to do it, create a new C# Window application in Microsoft Visual Studio, and place the ChartViewer component onto the form.



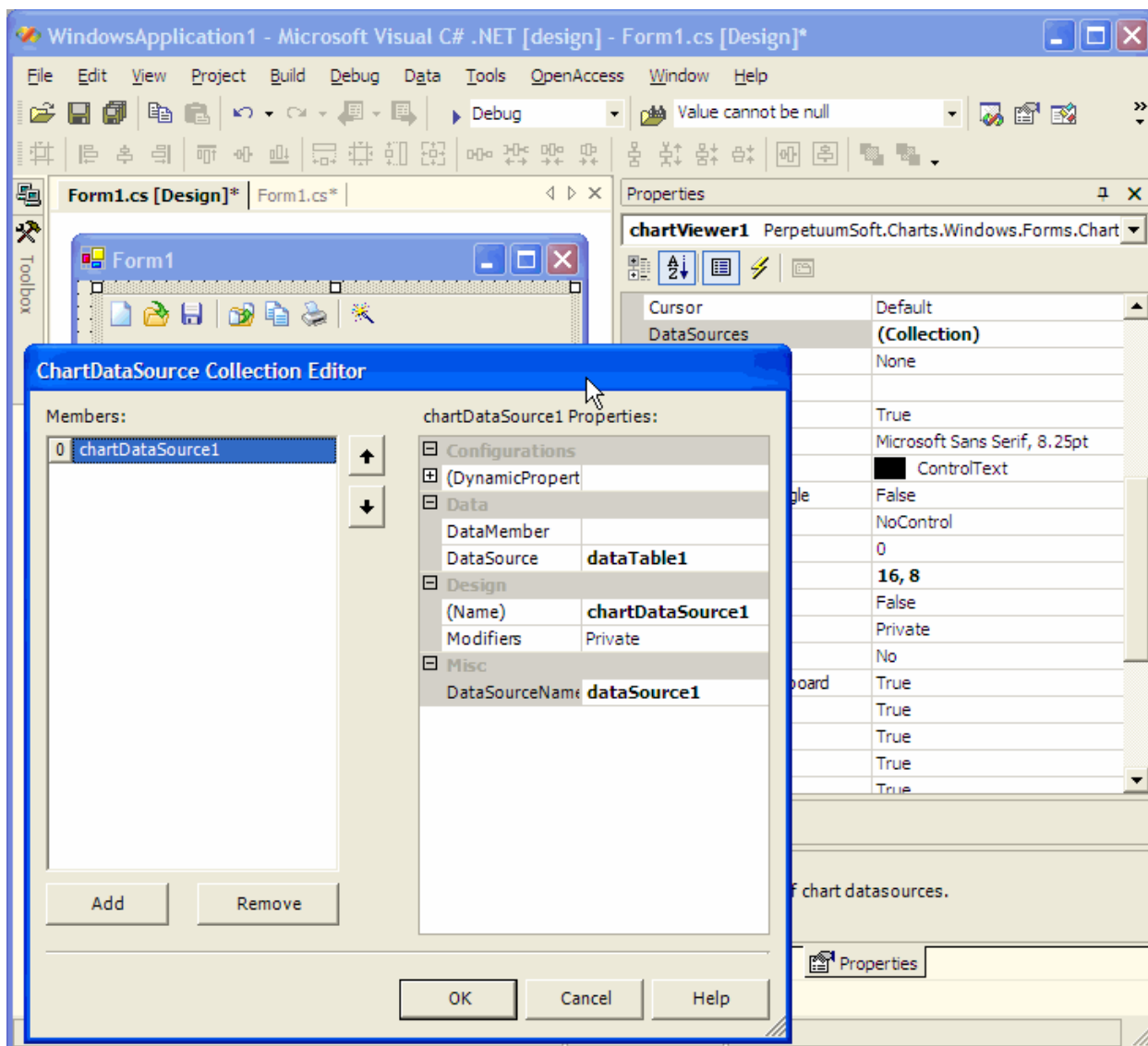
The task is to design a chart, which uses the DataSet table as a data source.

Place DataSet onto the form and create a table with a single column of double type in it. Add the following code, filling the table with data, when the form is loaded:

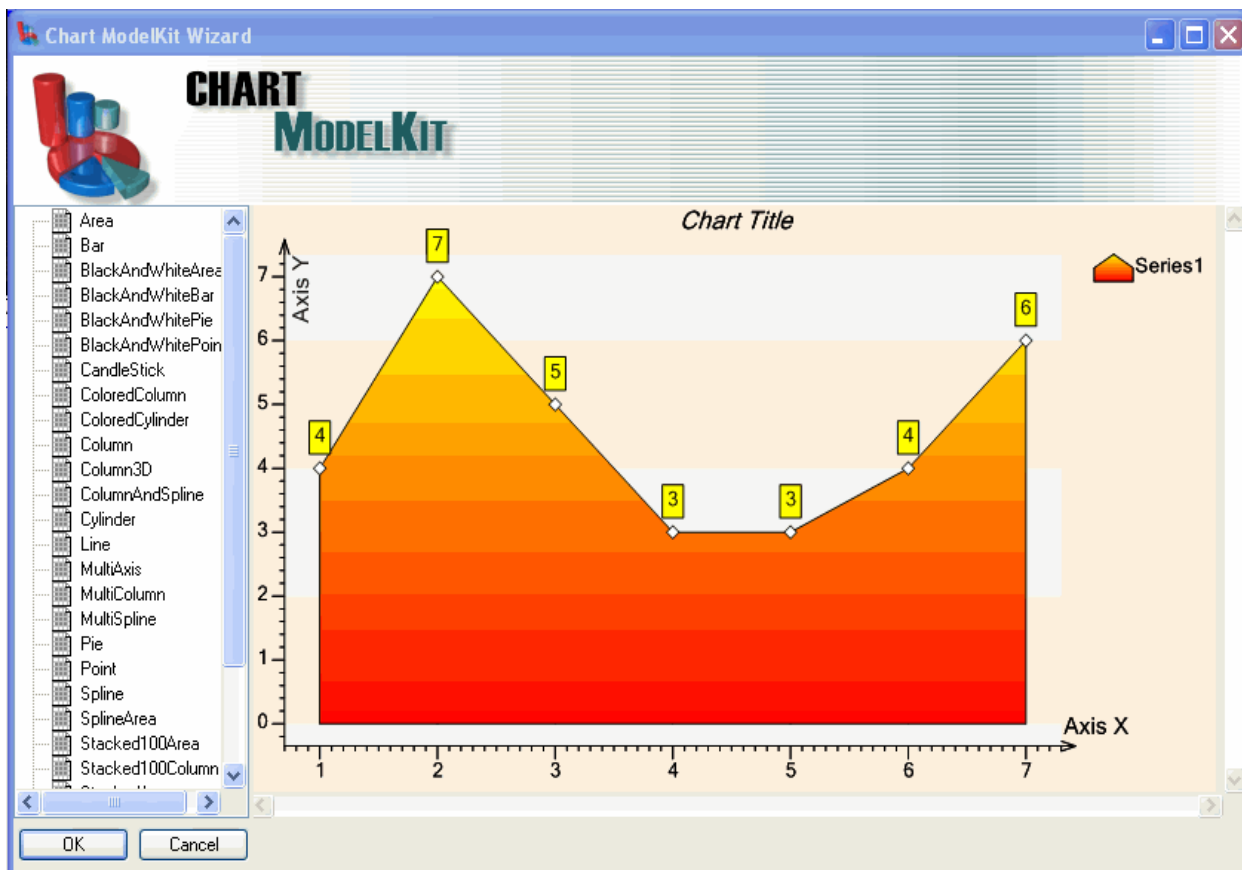
```
private void GettingStartedForm_Load(object sender, System.EventArgs e)
{
    dataTable1.Rows.Add(new object[] {7});
    dataTable1.Rows.Add(new object[] {4});
    dataTable1.Rows.Add(new object[] {9});
}
```

Add a new data source in the ChartViewer. Assign its name and data source (the DataSource property).

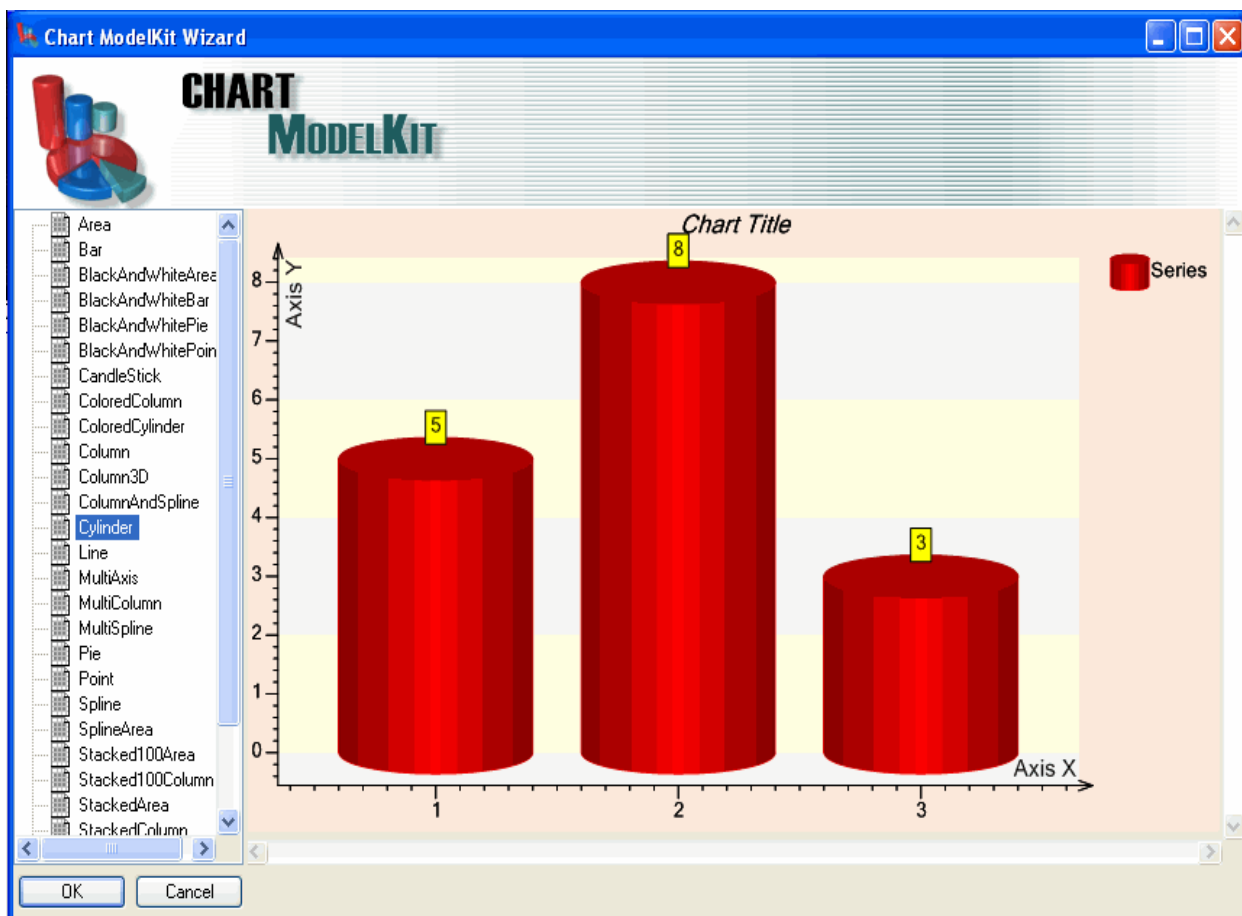




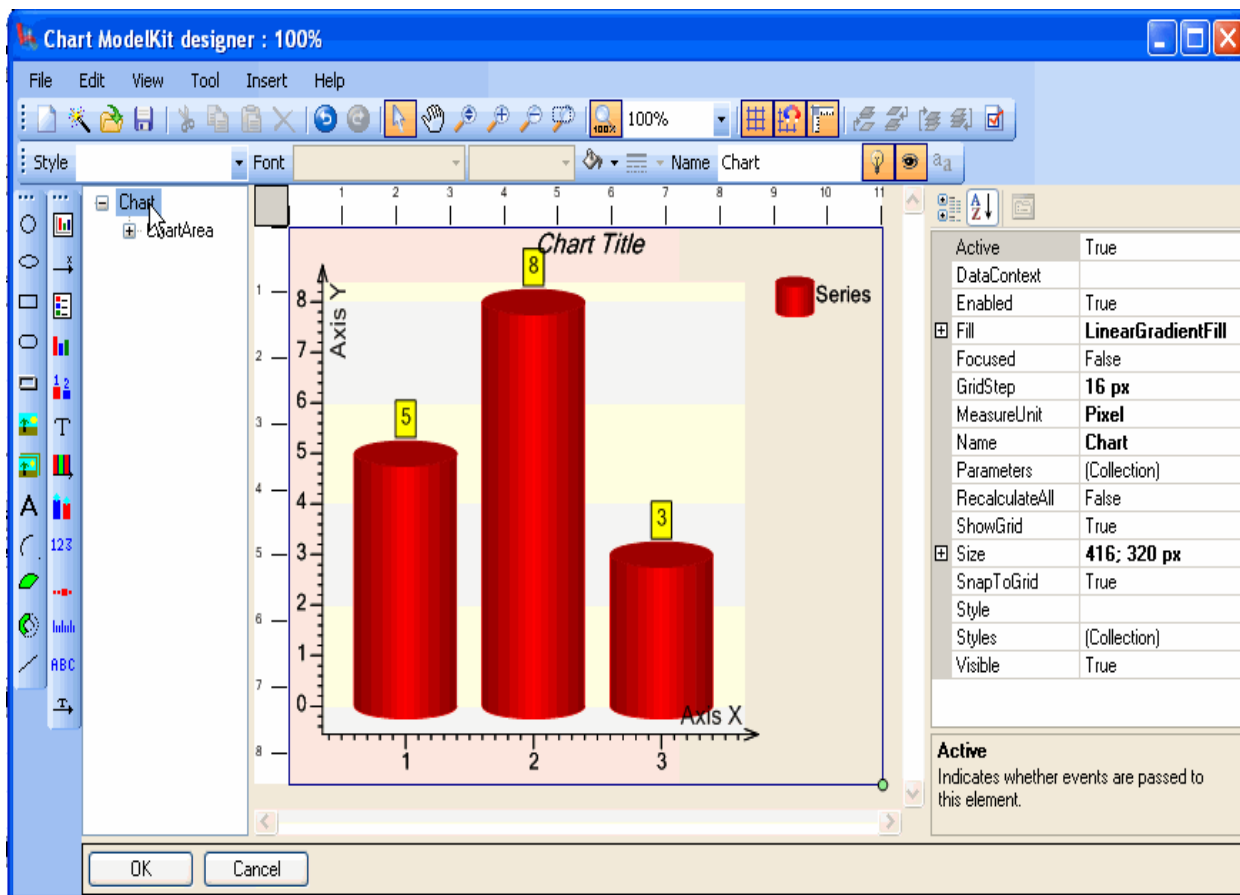
Double click on the ChartViewer component to run the Designer. When you open the designer the Wizard window also opens. Here you can select a chart from a number of existing templates.



Select the chart from the list.













Click the "OK" button. The selected chart will be loaded to the designer.





The designer provides fast visual customization of the chart and all its elements.

The left part of the window contains elements intended for building the chart. These elements are described below.

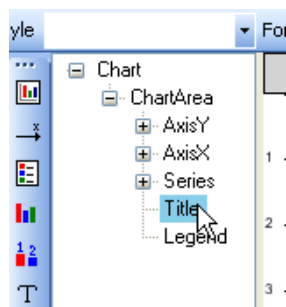
- ChartArea  is an element, containing all other chart elements: axes, series, titles etc. A chart can contain one or multiple ChartAreas.
- Axis . Values of any type series (except for the Pie Chart) are laid along the axes. To create the chart correctly, the ChartArea should contain two axes of different directions: X-direction and Y-direction.
- Series  represents a single set of data and contains properties, defining the way this data is displayed. The display mode is assigned by the SeriesView property. Data used for building a series can be got from a data source or can be assigned directly in a series.
- Legend  is intended for displaying a chart legend.
- ValueLabels  is used for displaying values of every point on the chart.
- Title  is intended for assigning the chart title.
- ScaleLabels  is used for displaying values on the axes.
- CustomLabels  is used for displaying arbitrary text labels on the axes.
- GridLines  is intended for displaying a grid on the axis.
- Ticks  is used for assigning ticks on the scale.



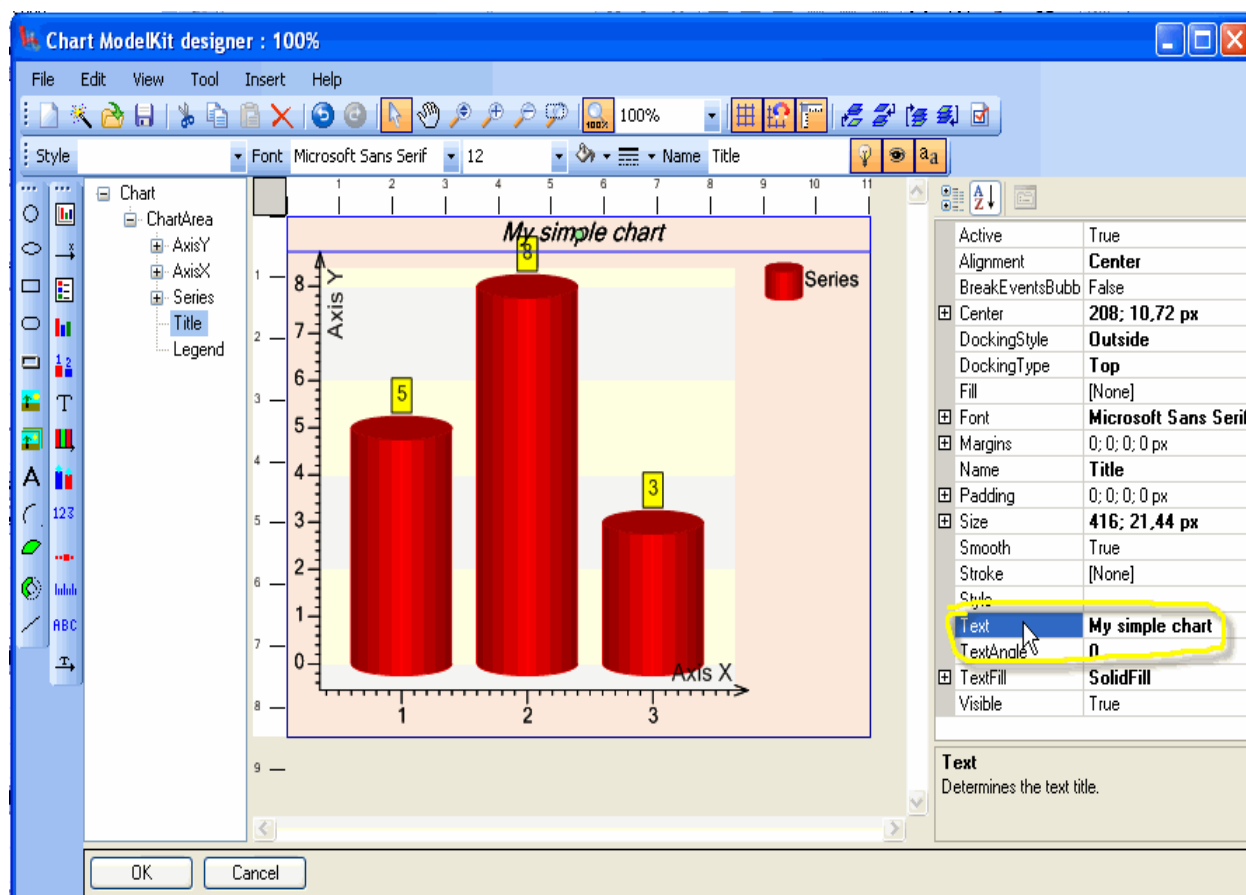
- ScaleMarks  is used to place markers on the axis.
- ScaleTitle  is used for assigning the axis title.

The chart, created with the help of the wizard, contains several values, so it is possible to assign chart appearance without binding it to a data source.

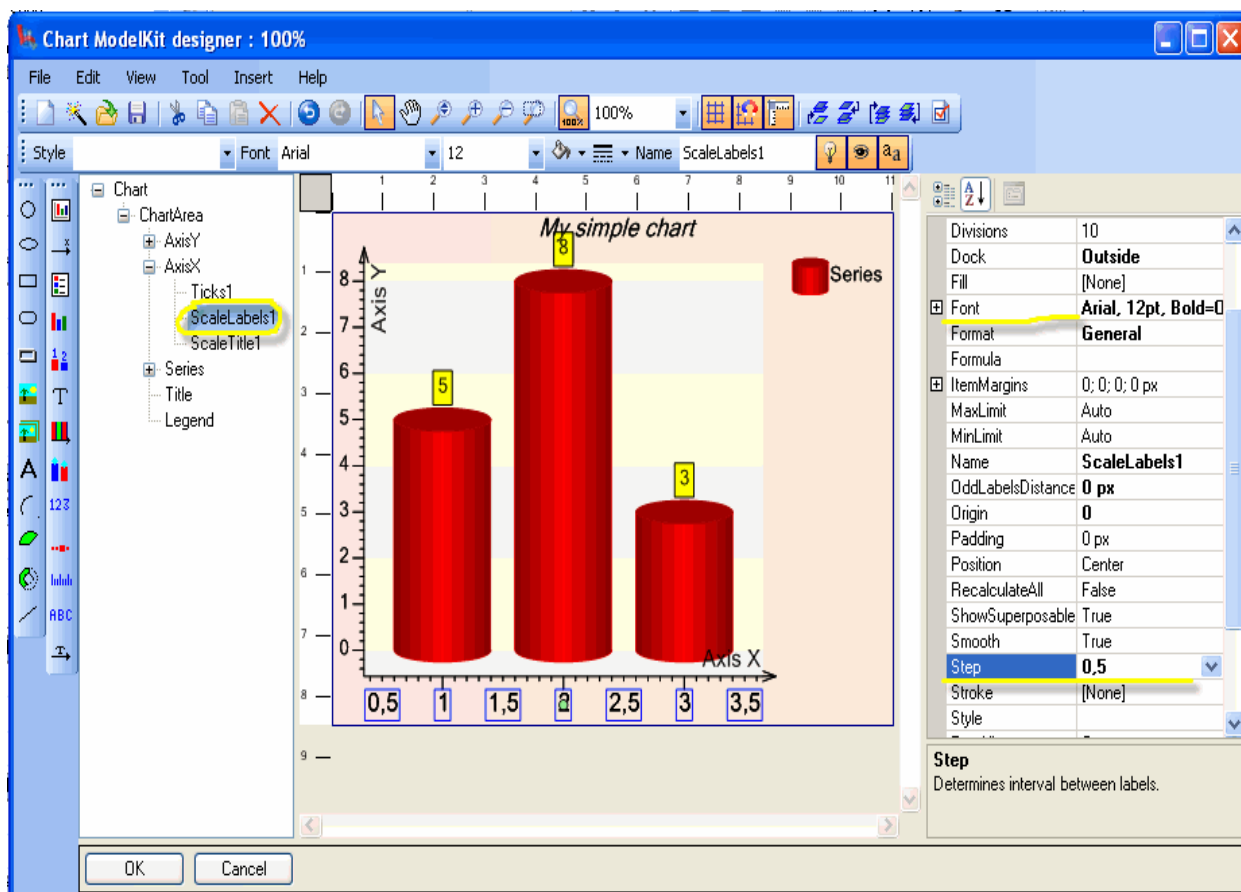
Let's customize chart appearance. Change the chart Title text. To do it, select the Title object in the chart tree.



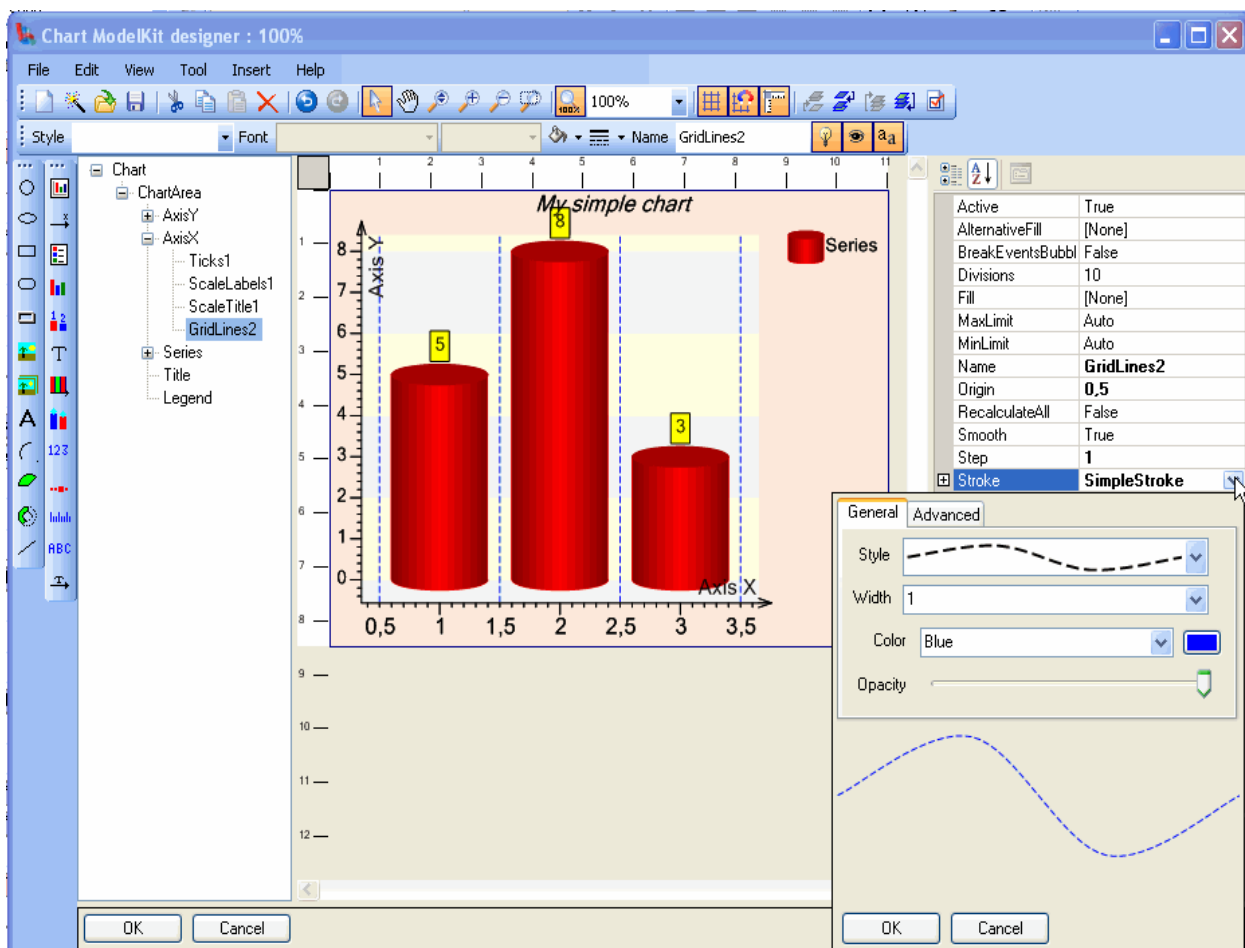
It is also possible to select necessary object by clicking on it in the designer. The selected element will be highlighted in the chart tree and all its available properties will be displayed in the property grid on the right of the designer. Change the Text property of the Title object:



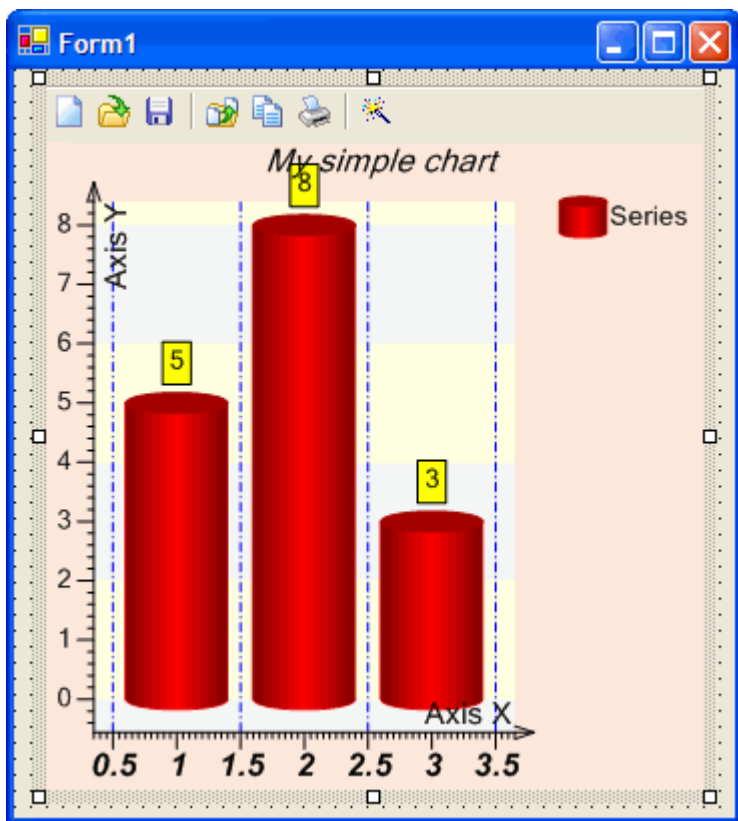
Select labels on the X-axis with a mouse and change some properties, for example font and step. Please pay attention to the fact that the ScaleLabels is located not in ChartArea but in Axis.




Change the Step property of the Ticks element on the X-axis. Add the GridLines object to the X-axis, and assign its Stroke, Step and Origin properties.

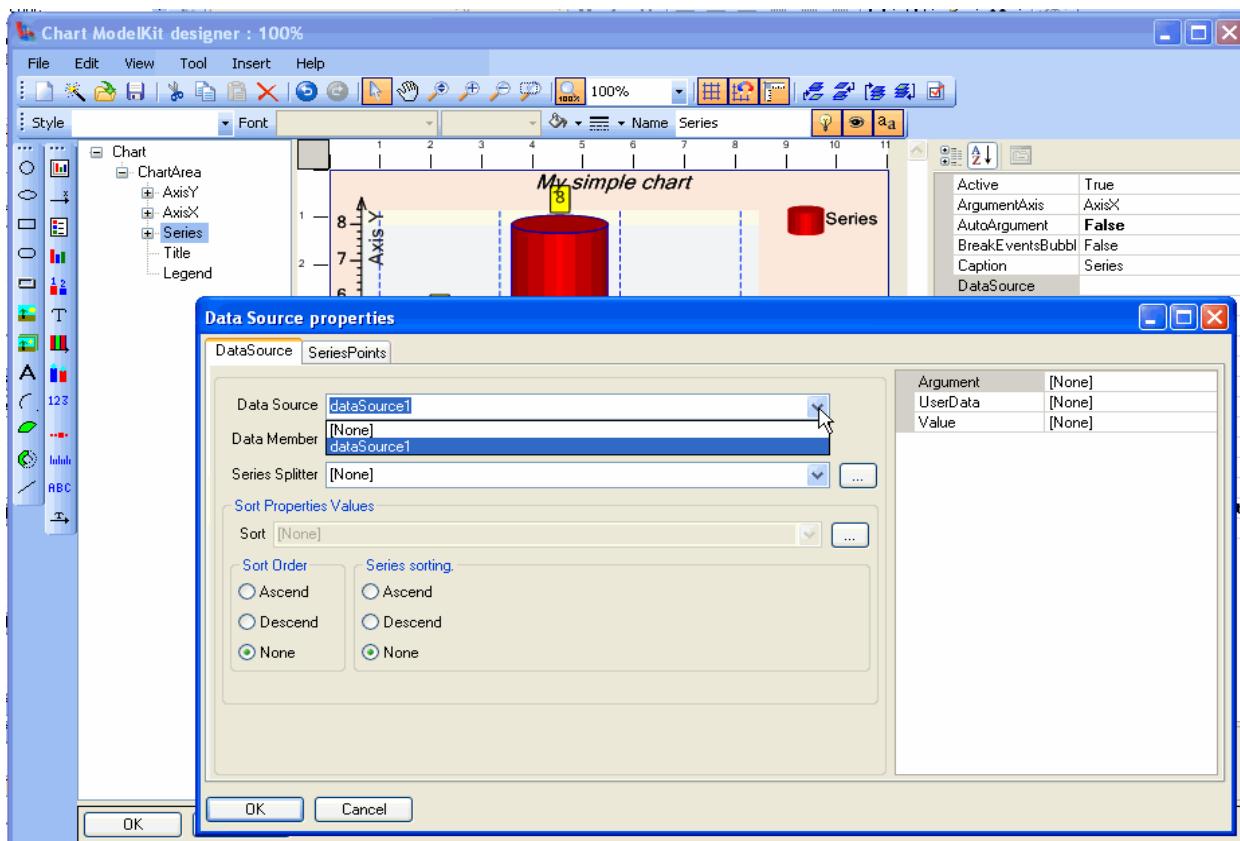


Save the created chart to a file and close the designer by clicking the "OK" button.

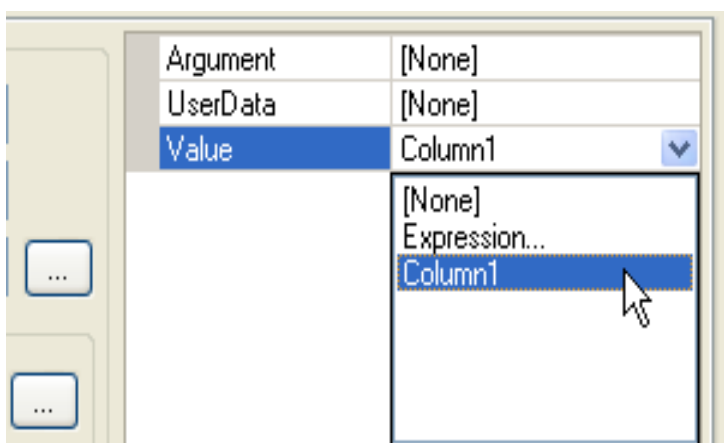




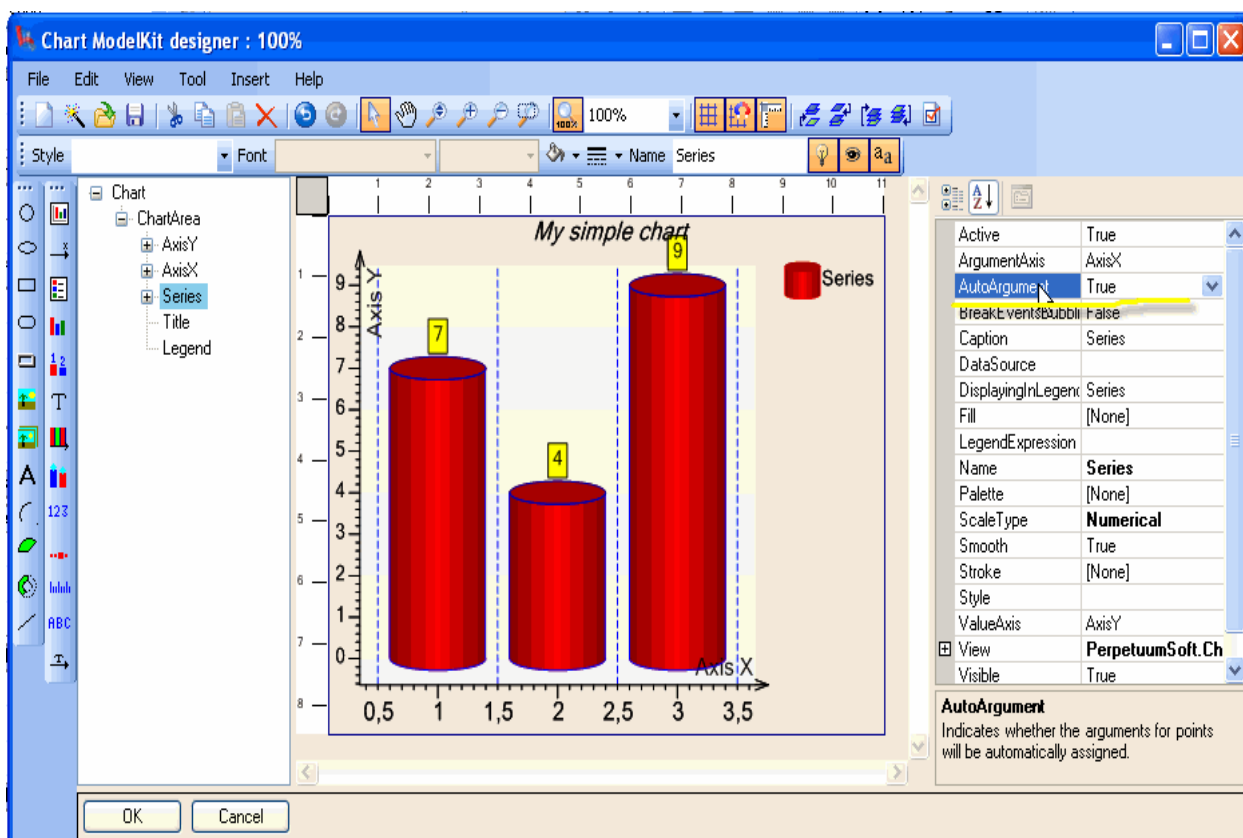
Let us build a chart with the use of data from the table. Run the created chart and with the help of the  tool bar button open the designer window. Select the series object by clicking on it with a mouse. Select the DataSource property and open a window for editing chart data source properties.



Select the dataSource object as a chart data source and bind Column1 to the series Value field.



Click the "OK" button. In the series, change the AutoArgument property to true, thus you specify that arguments will be automatically substituted at every chart point.



If the specified data source contained arguments together with values, it would be possible to assign binding of this field to the series Argument field.

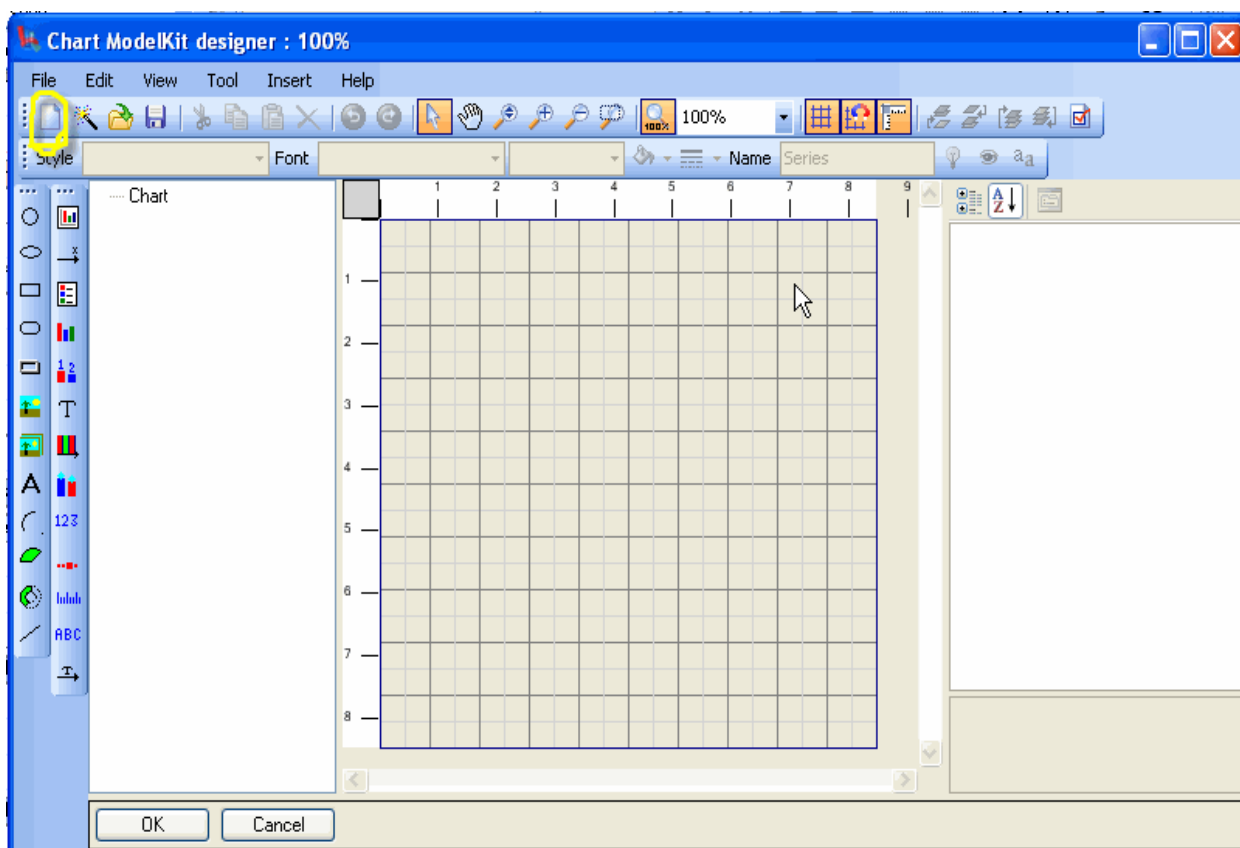
Save the modified chart to a file and then load it to the ChartViewer at the design-time.

This example illustrates the ability to build charts with the help of wizard. But it is not the only way of creating charts. Let us consider designing the chart without using wizard.

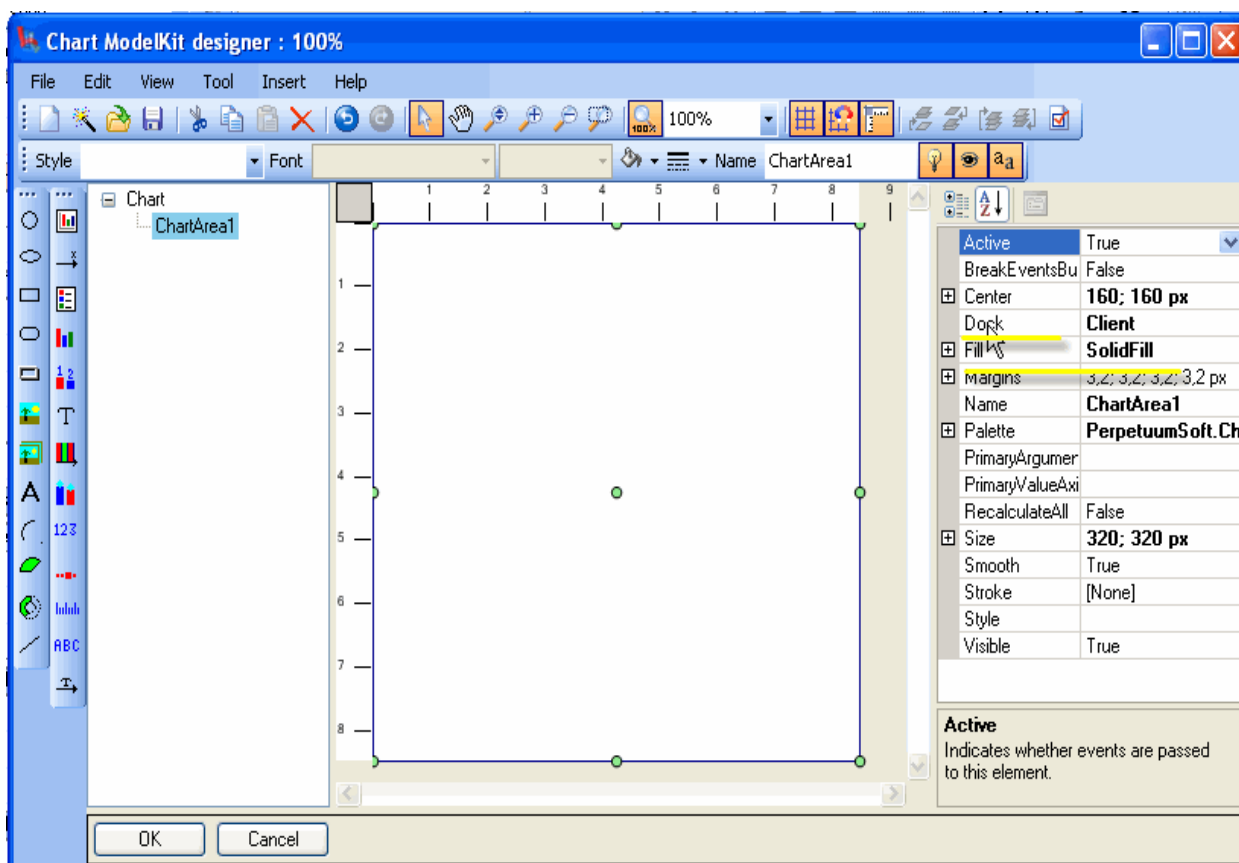


## Creation of a Simple Chart without Using Wizard

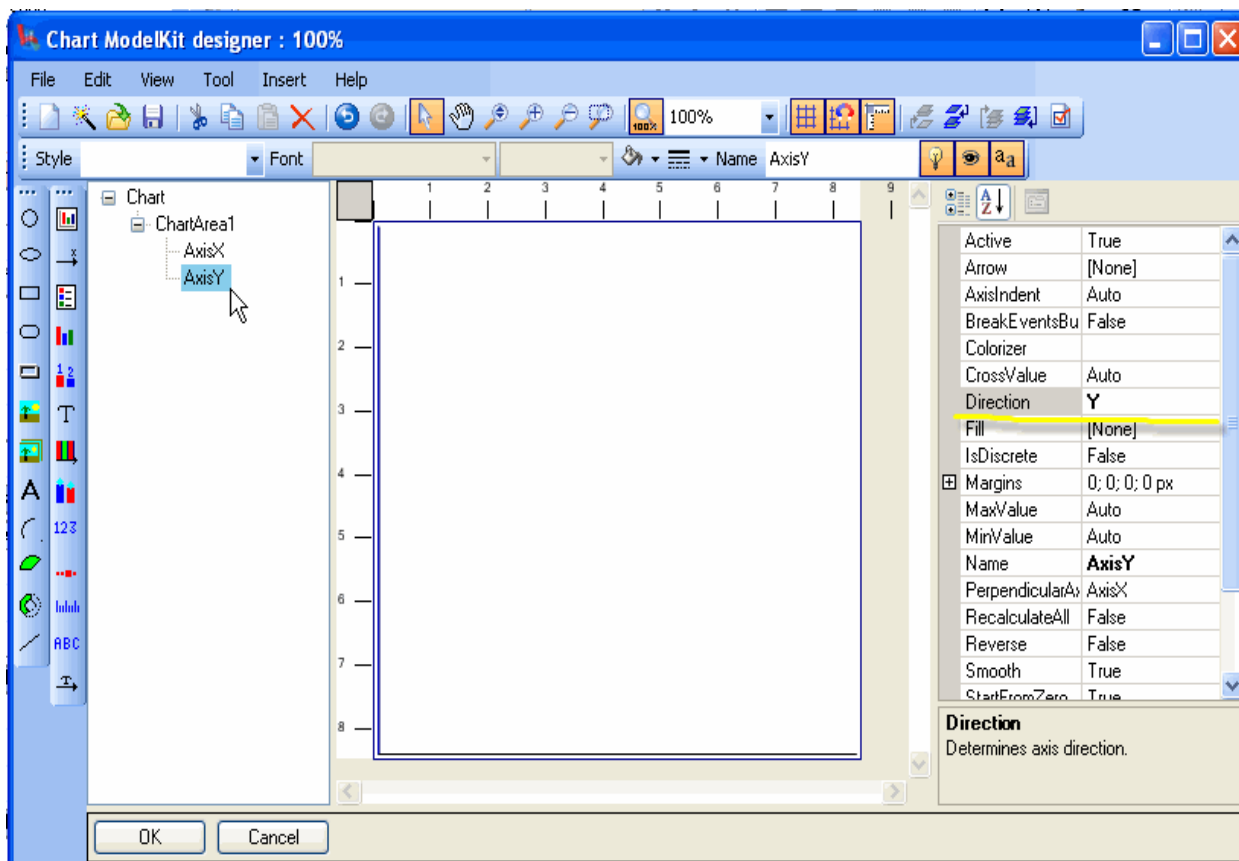
Double click on the ChartViewer component, open the designer window and create a new chart.



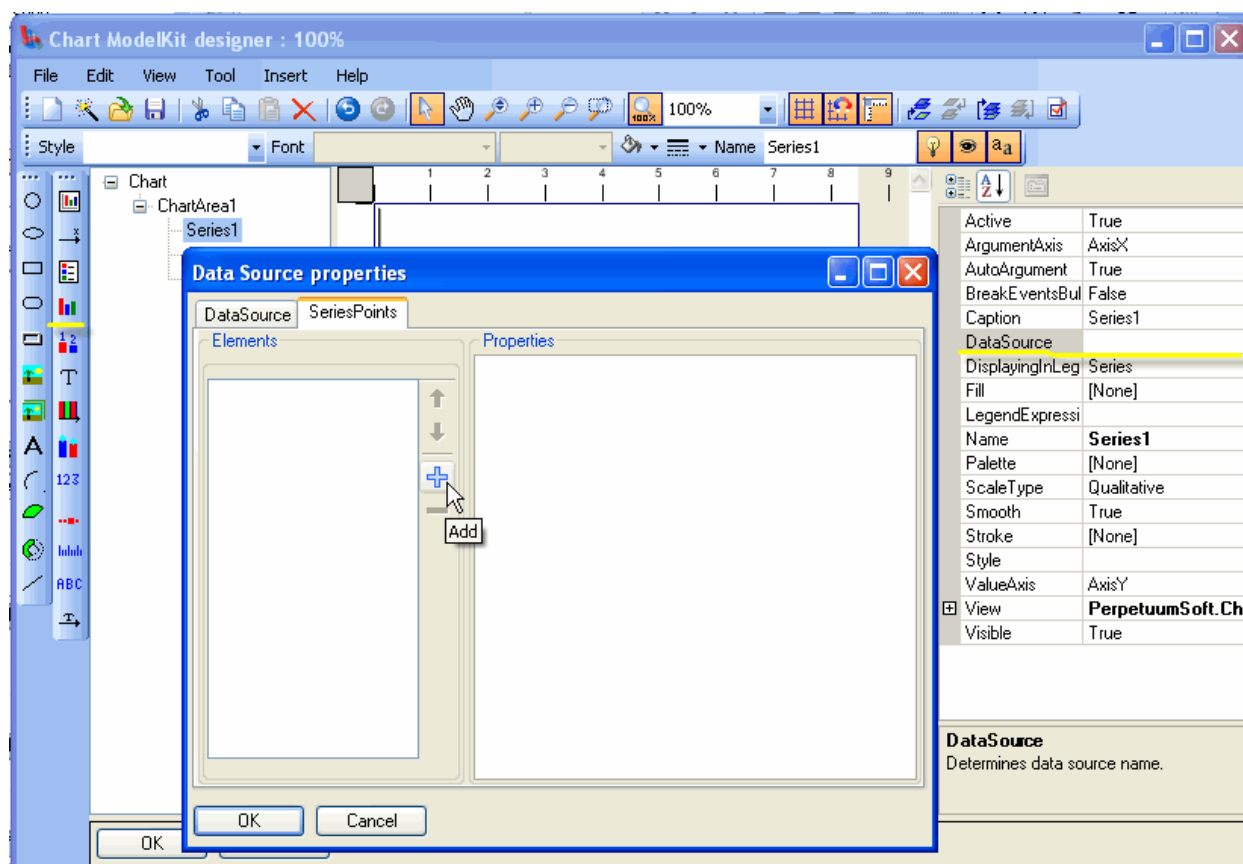
Place the ChartArea element to the chart, set the Fill and Stroke properties responsible for the object fill and frame. Set the Dock property to Client, so the ChartArea element will be automatically expanded to fit the whole work area.



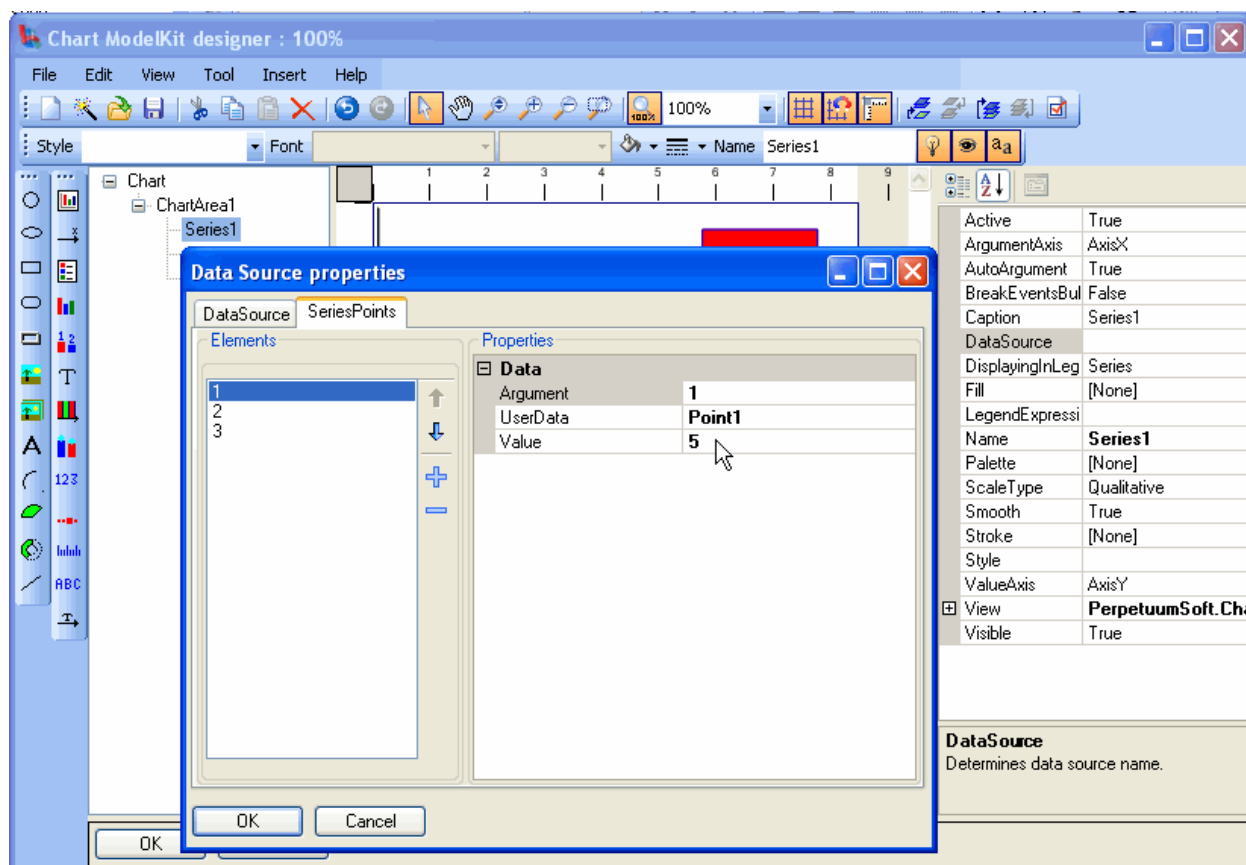
Place two axes to the ChartArea and change direction of one axis by setting the Direction property to Y.



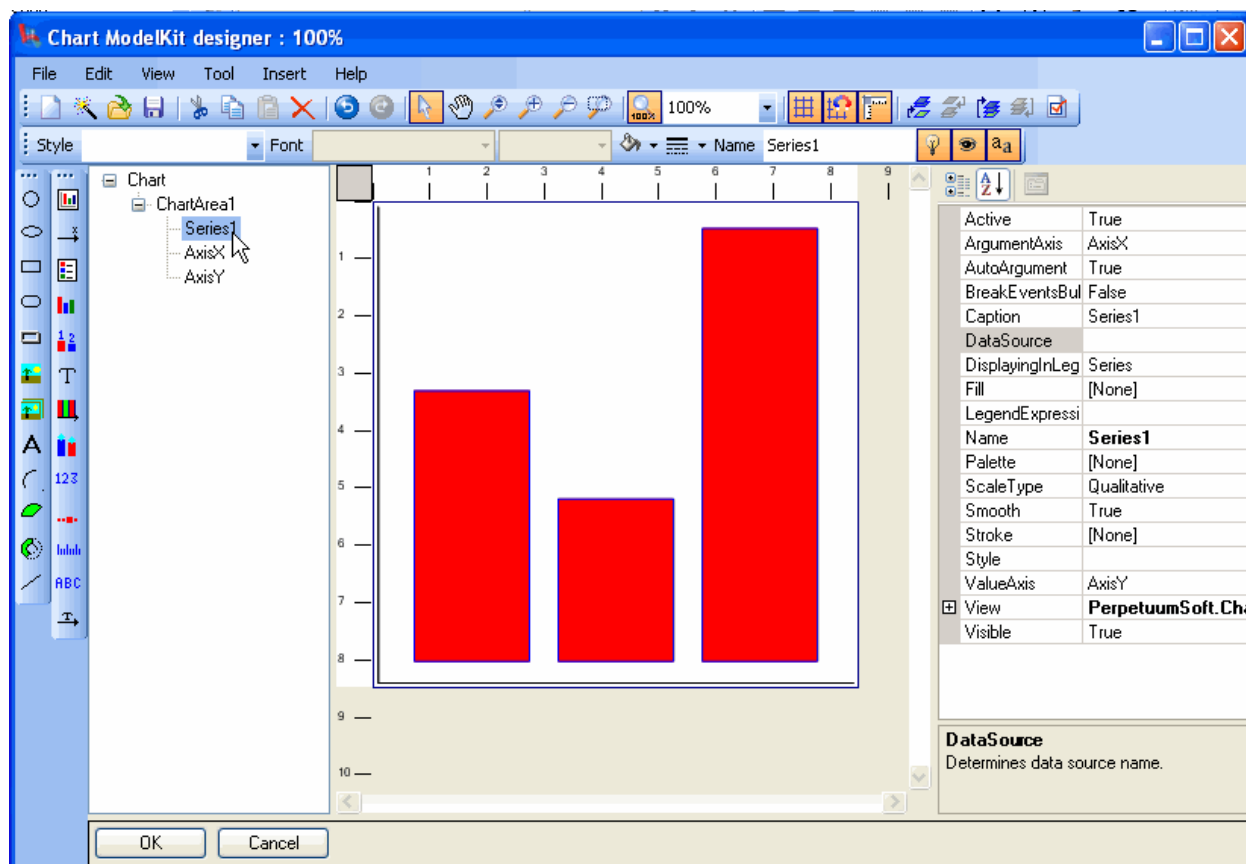
Add the Series element to the ChartArea. For a start, assign data directly in the series without using a data source. To do it, open the DataSource property editor and select the DataPoints tab.



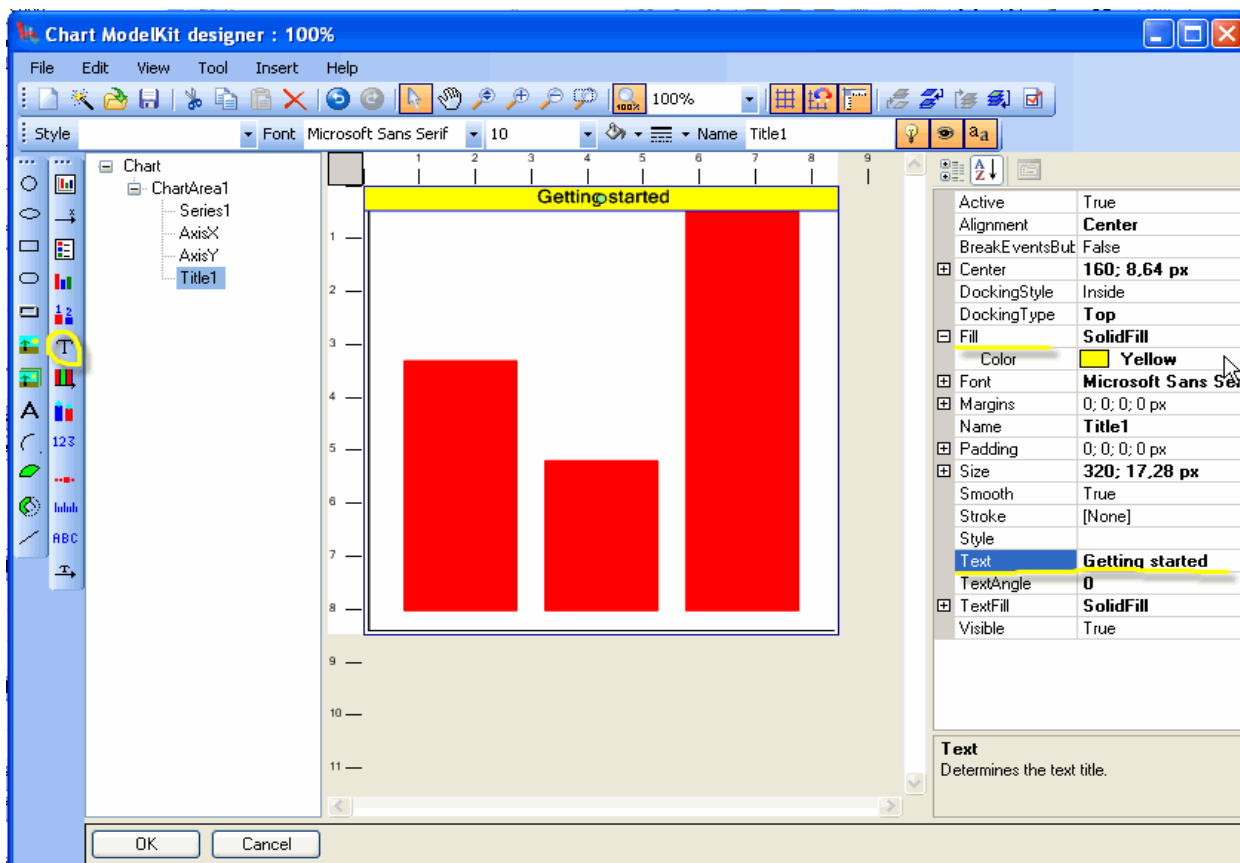
Add several values to the collection. It is not necessary to fill the UserData field. This field can be used to output information not bound to value in the ValueLabels element.



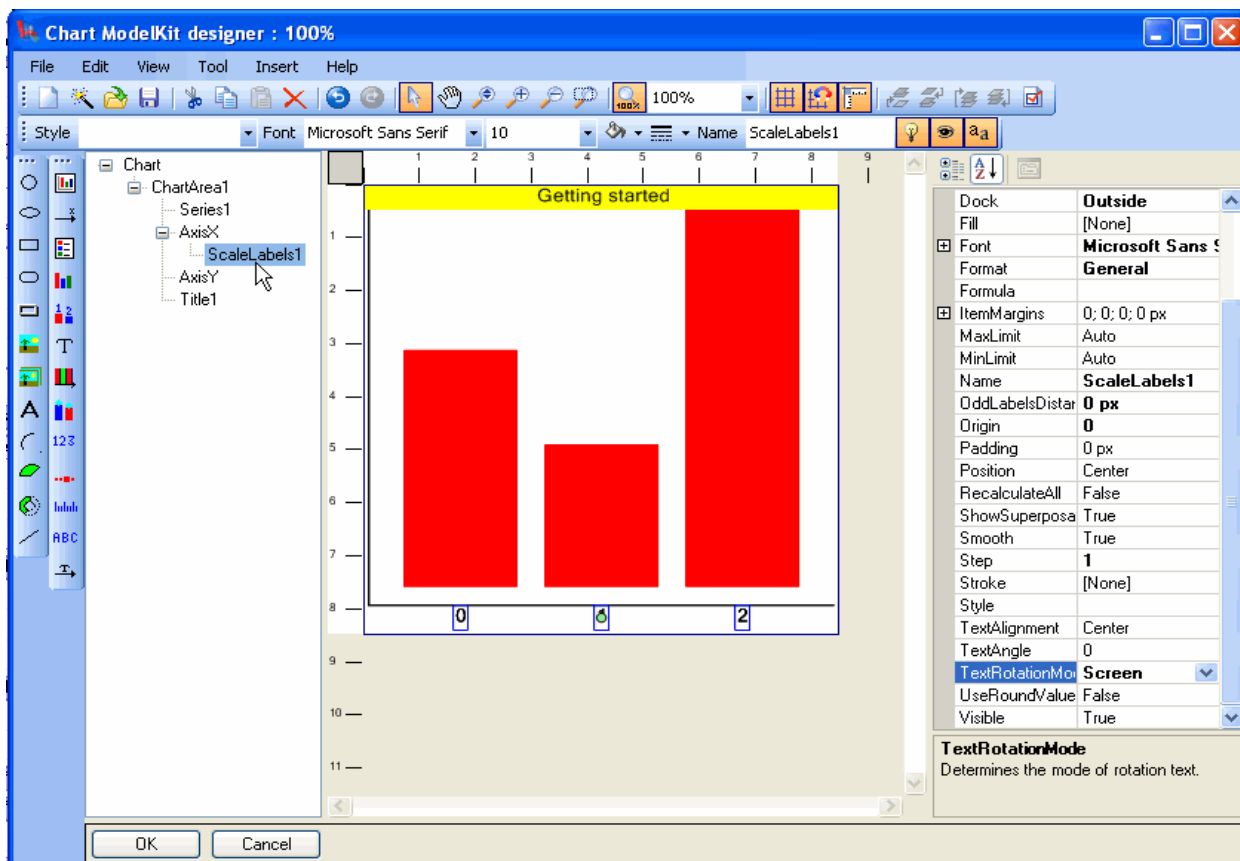
After that, press the "OK" button and the chart displayed in the image below will be generated.



Now let us assign chart appearance. Add the Title element to the ChartArea and customize its appearance.

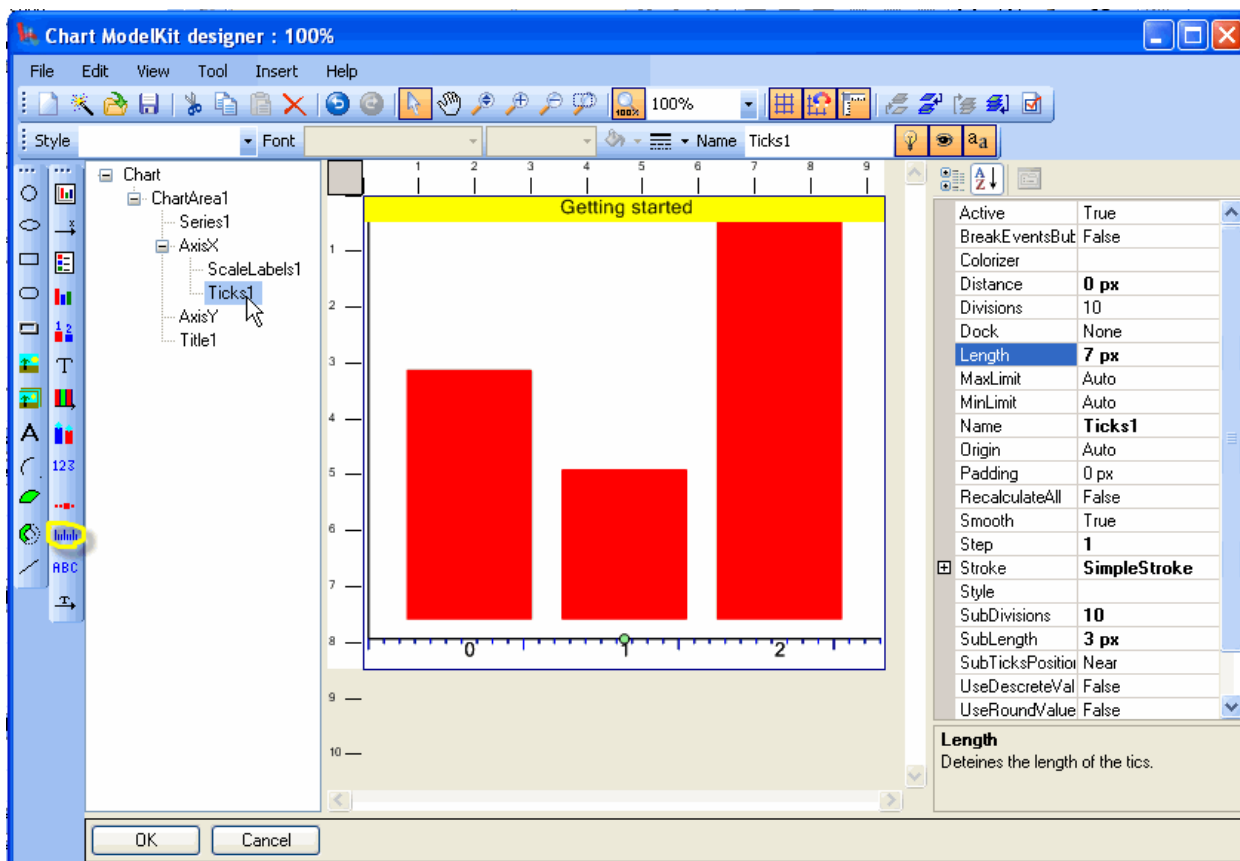


Set axes appearance. Assign numeric values on the axis with the help of the ScaleLabels element. Place ScaleLabels to the X-axis, assign their origin and step. Set the docking property for automatic assignment of the axes size subject to ScaleLabels.

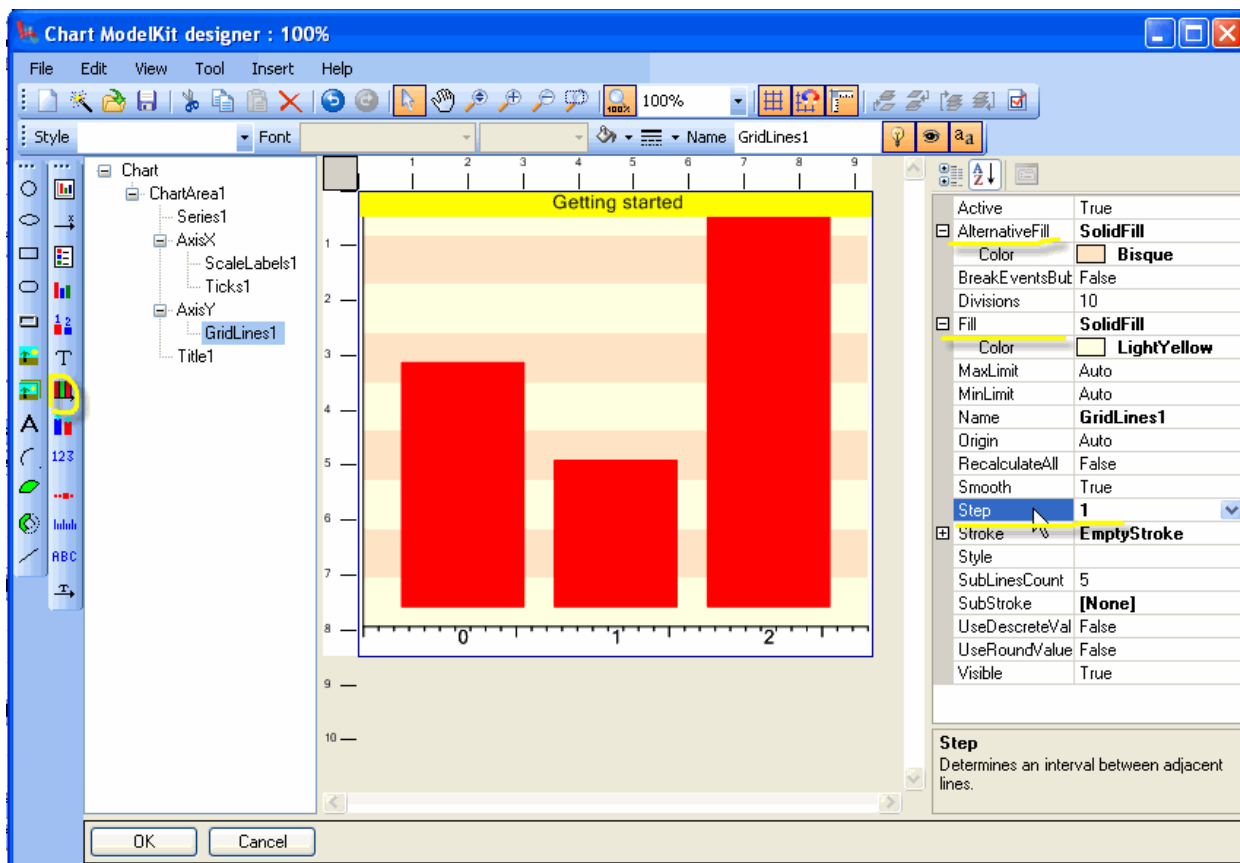




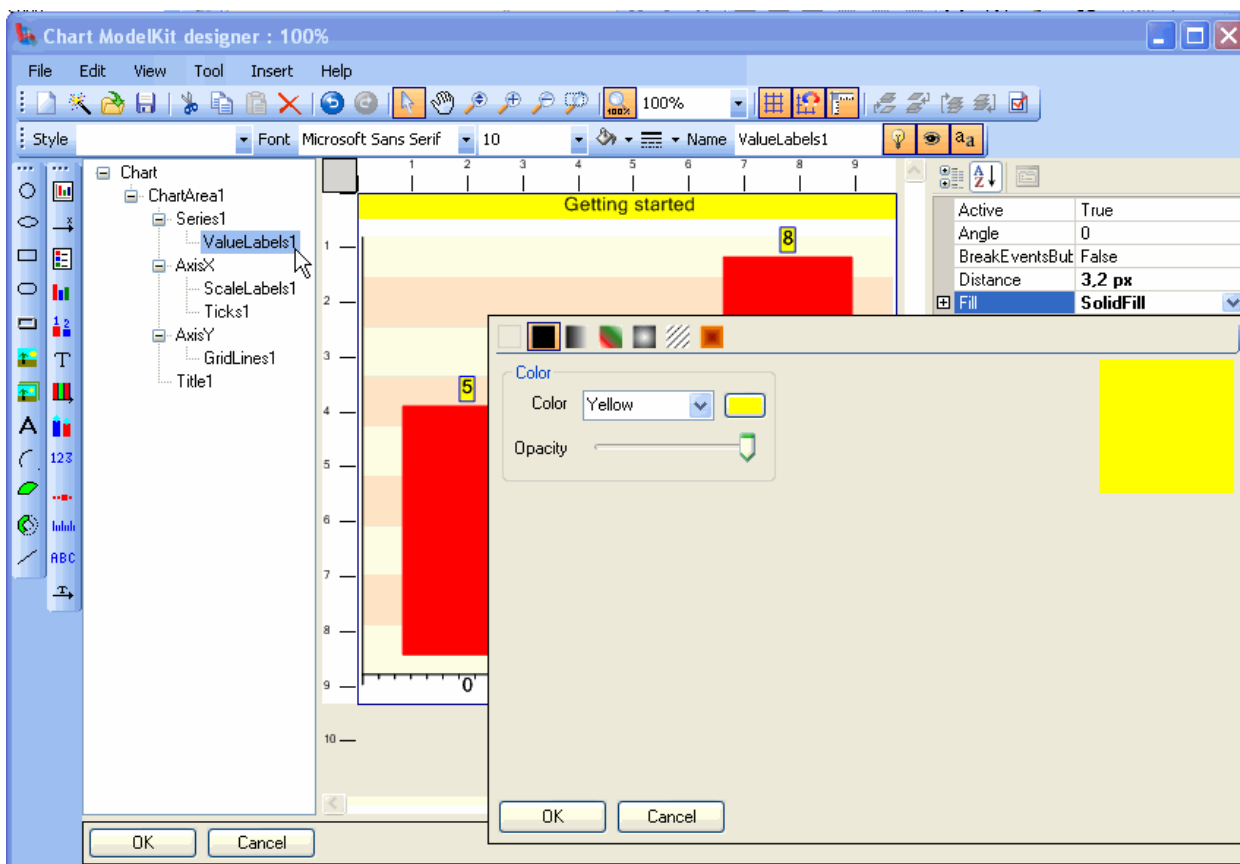
Add the Ticks element and customize it analogously to ScaleLabels.



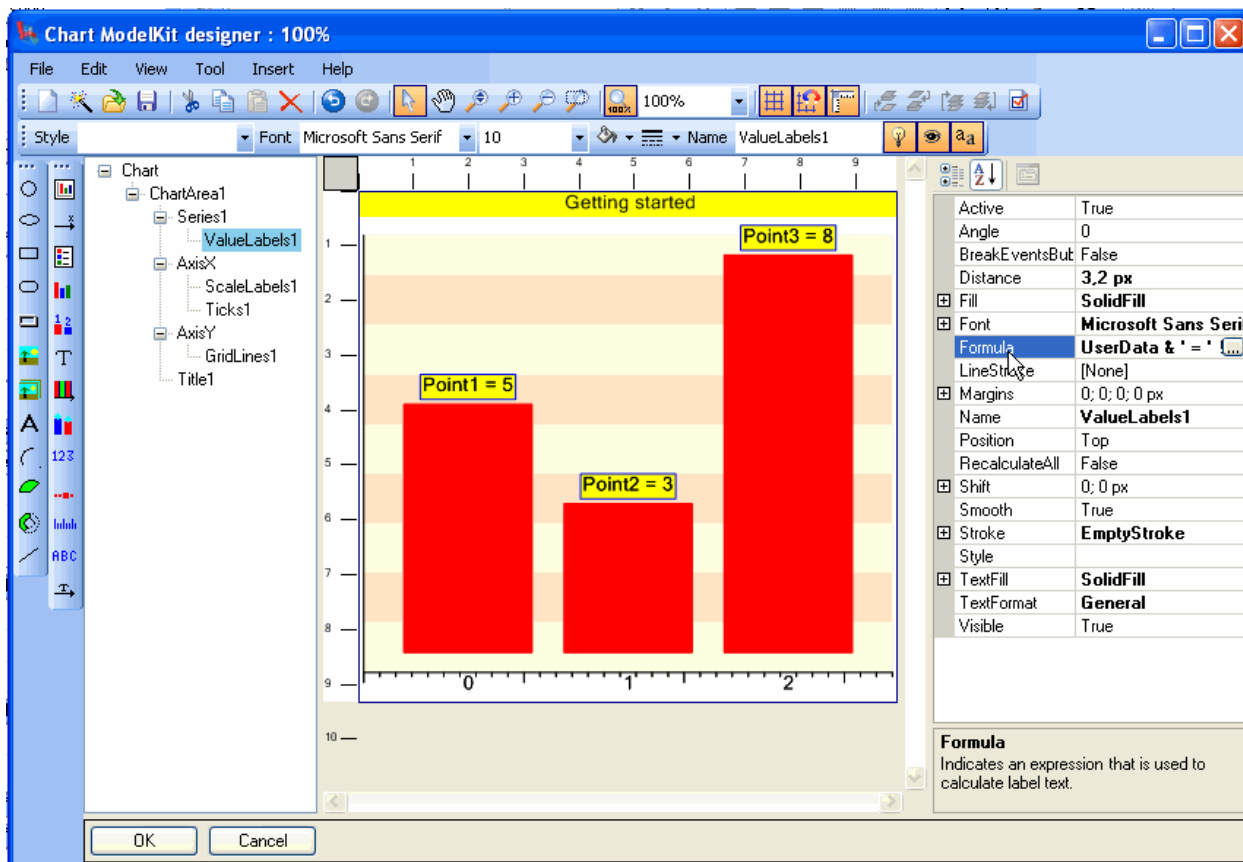
Place the ScaleLabels and Ticks elements onto the Y-axis and set them analogously to X-axis elements. Place the GridLines element onto the Y-axis set the Step and Origin properties similarly to Ticks and ScaleLabels. With the help of the Fill and AlternativeFill properties assign fill of the alternate strips.



Place the ValueLabels element to the series. It will display every value in the chart by default.

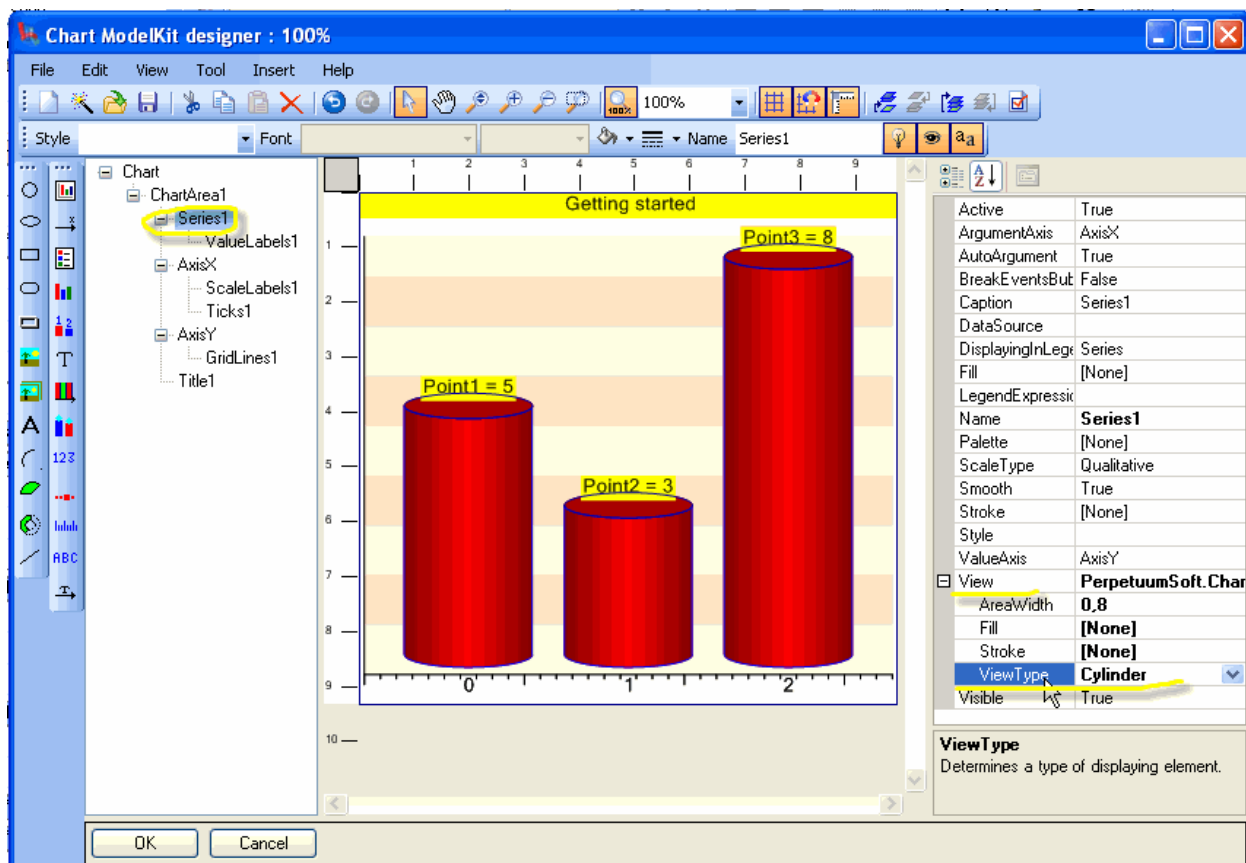


It is possible to specify formula in the ValueLabels Formula object field. With the help of this formula the output value will be assigned. Set this property in the following way: `UserData & ' = ' & Value`.

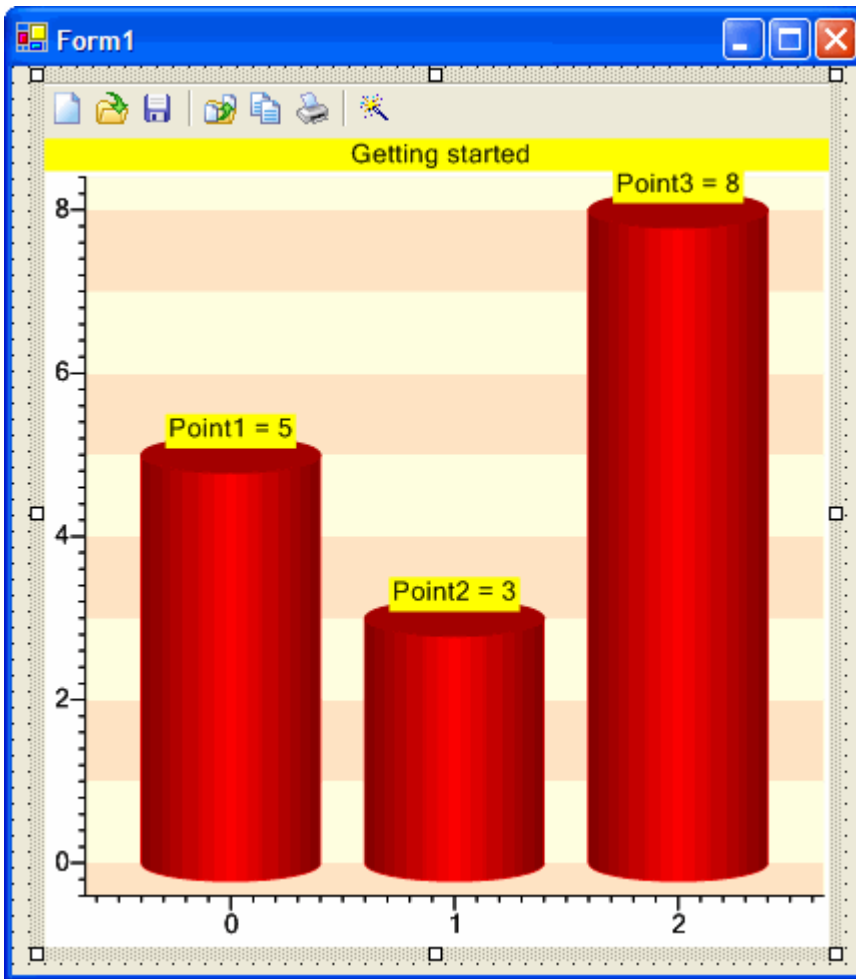


Change the type of the displayed object from rectangle to cylinder in the ViewType property.





Simple chart is designed. Click the "OK" button and close the Designer.

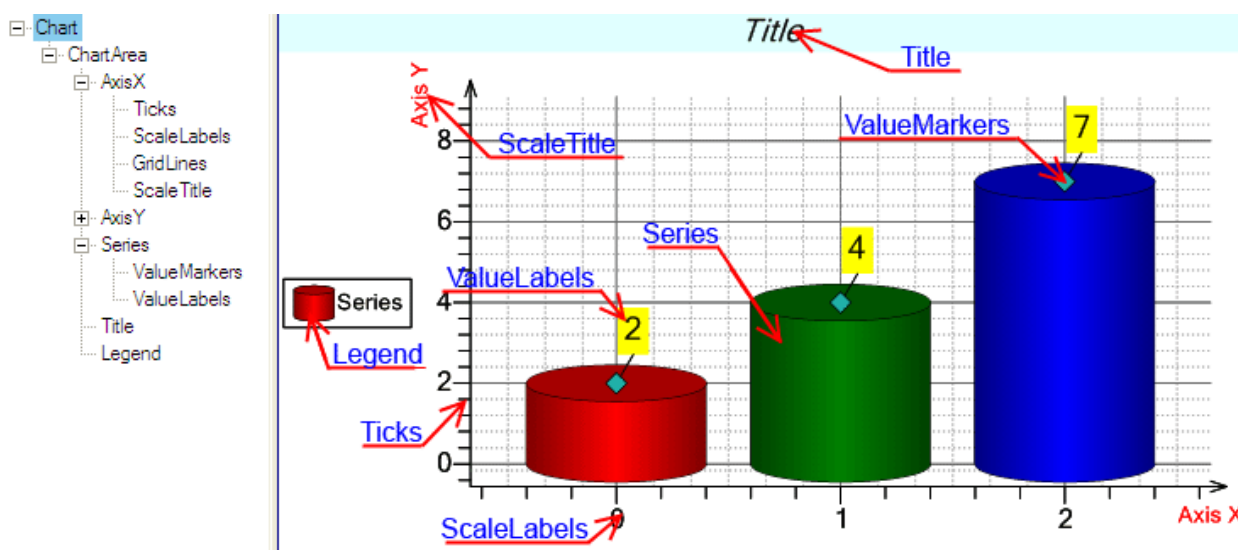


The GettingStarted application is included in the delivery package.

## Chart Model

Chart is a compound object, having hierarchical tree-type structure. The whole chart consists of separate simple elements. Chart decomposition into separate elements provides the ability to create charts of different types, due to variety of elements combinations. Setting of elements' properties became more intuitive and easy.

The following constituents are included in the chart: Series, displaying data; Axes, assigning chart metric; various Labels on axes and series; Legends, displaying notation conventions, used in chart; other decorative chart elements, used for chart design. All these chart elements are represented in the SharpShooter Charts as separate elements with various properties. The chart can combine elements in arbitrary amount. But usually, the chart has the following typical structure. The ChartArea is located in the chart. This element includes Axes, Series, Legend, and various explaining Labels, Titles, Images, etc.



Axis assigns direction and values range for the assigned direction. Ticks, ValueLabels, CustomLabels, GridLines, and ScaleTitle can be included in the axis, in order to visualize values.

Series is intended for displaying a data set. Depending on the assigned SeriesView, series can display data in different forms, for example, as bar chart, line chart, etc. ValueLabels and ValueMarkers, marking specific points, can be included in the series in order to visualize values.


Elements can be structurally subdivided into two groups: compound and simple. Compound elements can include other chart elements. Simple elements cannot include other chart elements.

## Description of Common Properties

Let's consider the Element class properties, peculiar to all descendant elements. The Name property assigns unique name of the element in the whole instrument. The Fill property assigns element's fill, which will be used when drawing this element. The Smooth property sets quality of the displayed graphics. The Visible property assigns elements' visibility. The Active property specifies if mouse events will be passed to the element. The BreakEventBubbling property specifies if mouse events will be passed to the parent element for further processing. The RecalculateAll property specifies if elements' expression will be recalculated when element's state is changed. If the RecalculateAll property is set to False, expressions will be recalculated only for the specified element and all elements, included in it, otherwise expressions will be recalculated for all instrument elements.

The process of receiving mouse events and their processing is conducted in the following way. The element receives events from the mouse, when it is located over this element. The element can process these events, and if the BreakEventBubbling property is set to False, it passes events to the parent. Thus, mouse events are passed from visual elements up the instrument tree.

## ChartArea

The ChartArea element  is a rectangular area for drawing the chart. It is a container for all chart elements. Position is defined by the Size and Center properties. The Dock property assigns type of ChartArea docking to the parent Chart object. If the Dock property value is set to ChartAreaDockStyle.None, ChartArea position doesn't depend on the position of the parent Chart and is assigned by the Size and Center properties. If the Dock property value is set to ChartAreaDockStyle.Client, position and size of the ChartArea will fit position and size of the parent Chart. The Margins property defines indents from the ChartArea boundaries. The PrimaryArgumentAxis and PrimaryValueAxis properties define argument axis and value axis that are used in series by default.

## Chart Series

The SharpShooter Charts supports the most popular Series types: Bar, Line, Pie, Stock, and many others. SharpShooter Charts concept provides separation of data storing and displaying, i.e. provides the ability to change series type without changing series data.

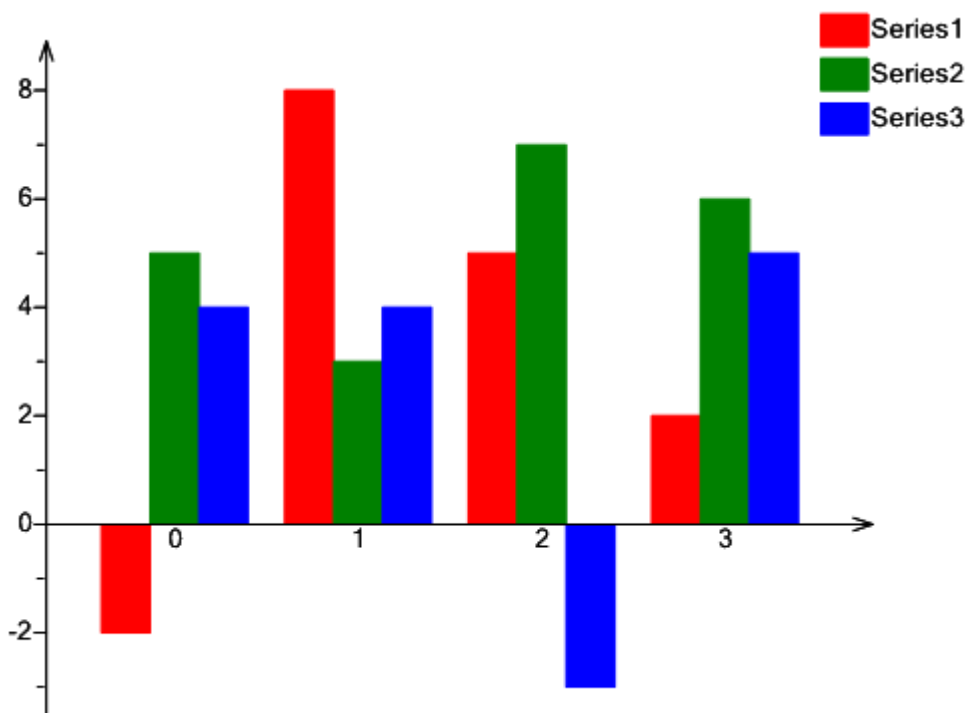
Series type is assigned by the View property.

Below you can find description of series types provided in the SharpShooter Charts.

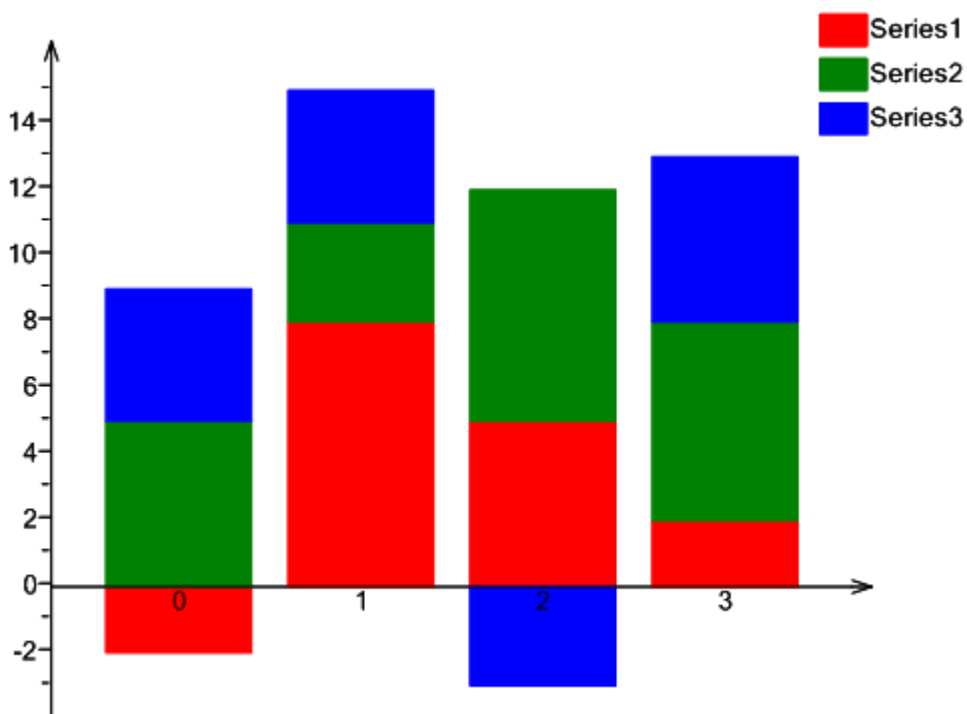
### Bar Chart

Bar chart is represented by the following classes: BarSeriesView, StackedBarSeriesView, and Stacked100BarSeriesView. The SeriesValue class is used as data type.

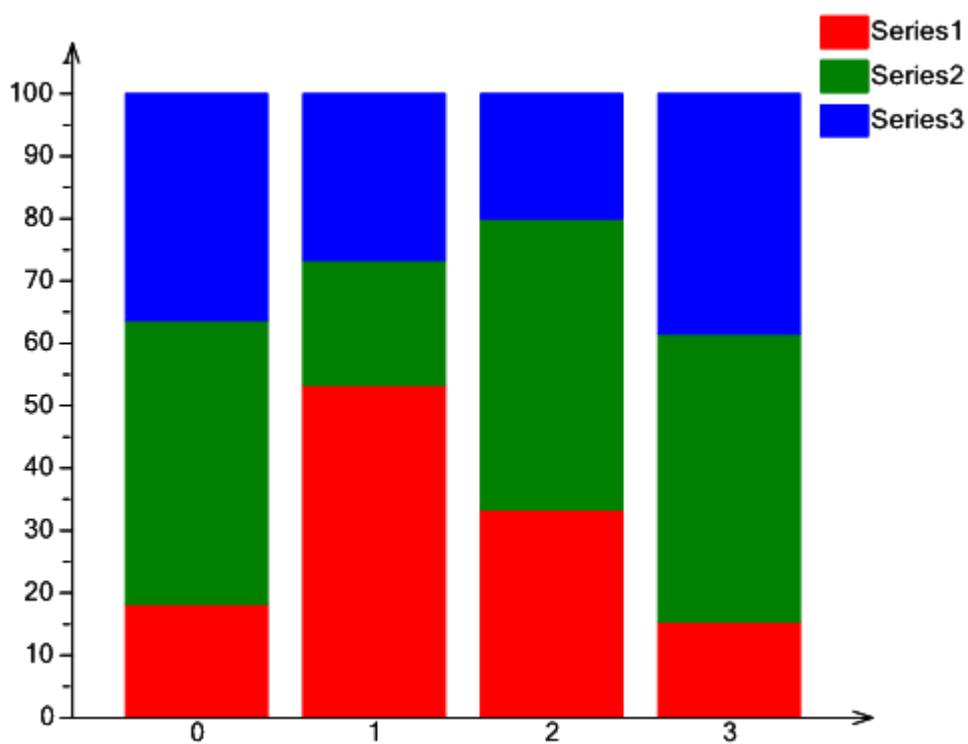
BarSeriesView is a bar chart, where bars, corresponding to different series values, are located side by side.



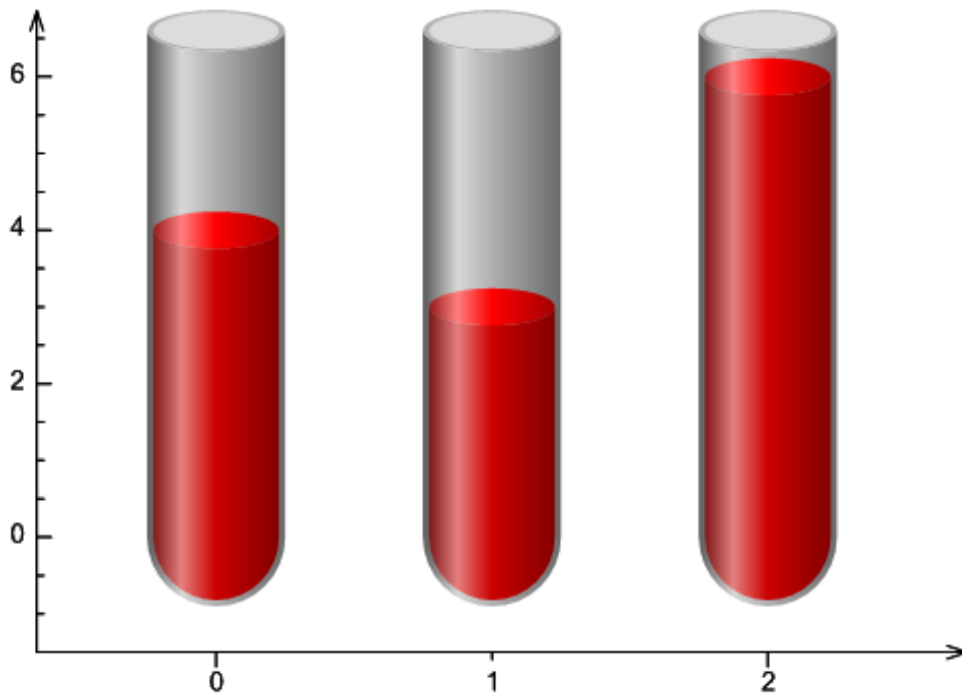
StackedBarSeriesView is a bar chart, where bars, corresponding to different series and a single argument, are located one over another.



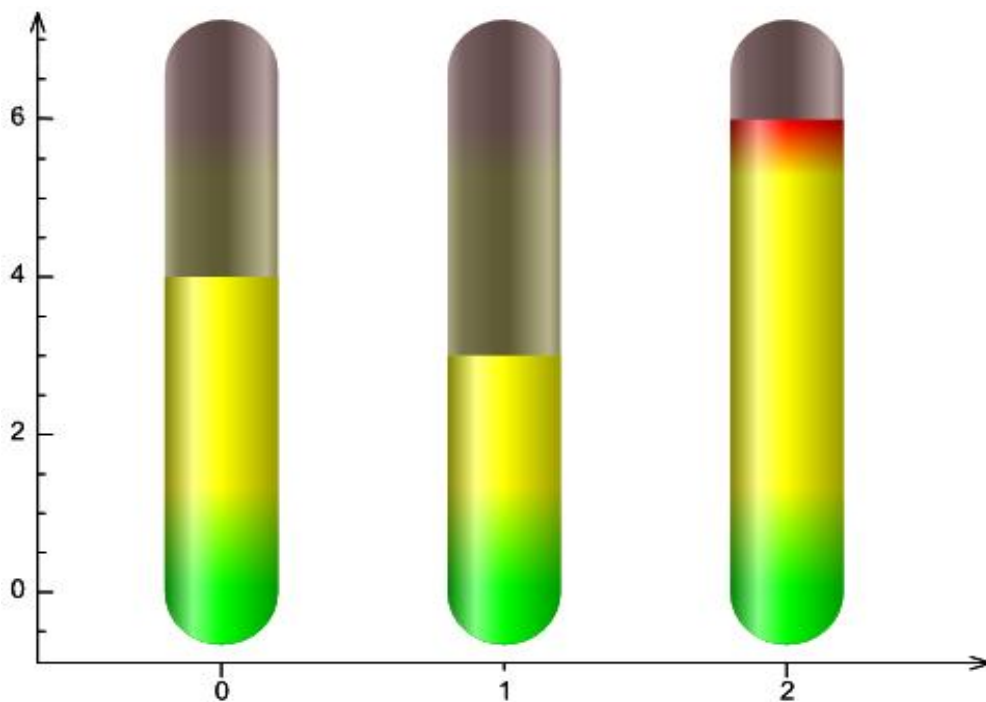
Stacked100BarSeriesView is a bar chart, where bars, corresponding to different series and a single argument, are located one over another, but they demonstrate relative value from the total amount of the given series type values at the given point.



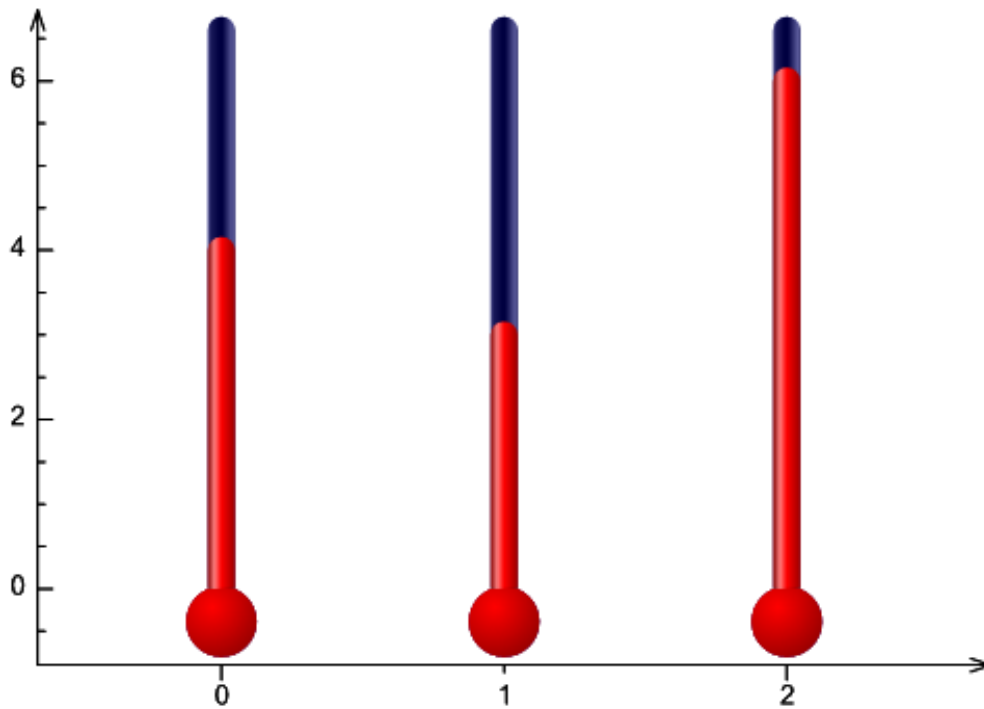
TankSeriesView is a bar chart which bars corresponding to different series values are located side by side and are drawn as Tank.



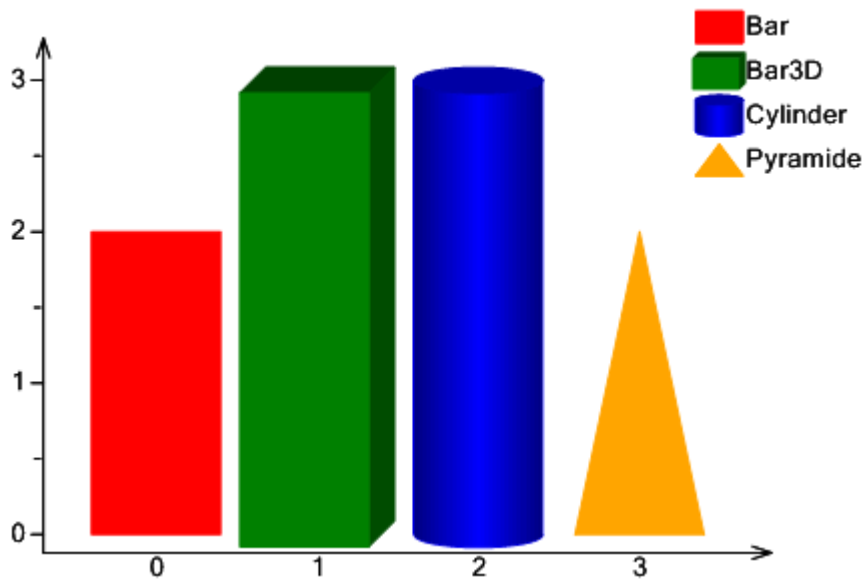
LevelSeriesView is a bar chart which bars corresponding to different series values are located side by side and are drawn as Levels. Upper and Lower layers are set with the help of the ActiveLayer and InactiveLayer properties.



ThermometerSeriesView is a bar chart which bars corresponding to different series values are located side by side and are drawn as Thermometers. Upper and Lower layers are set in the ActiveLayer and InactiveLayer properties.



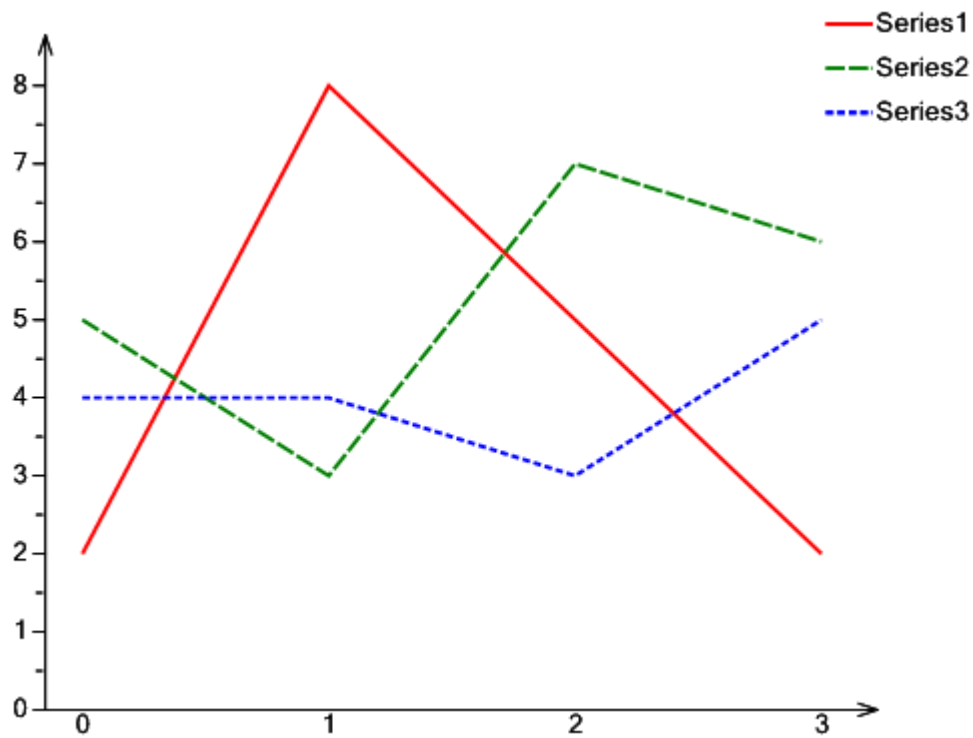
Bar charts allow assigning relative bar width (the AreaWidth property) and type of displayed elements (Bar, Cylinder, Bar3D, and Pyramide).



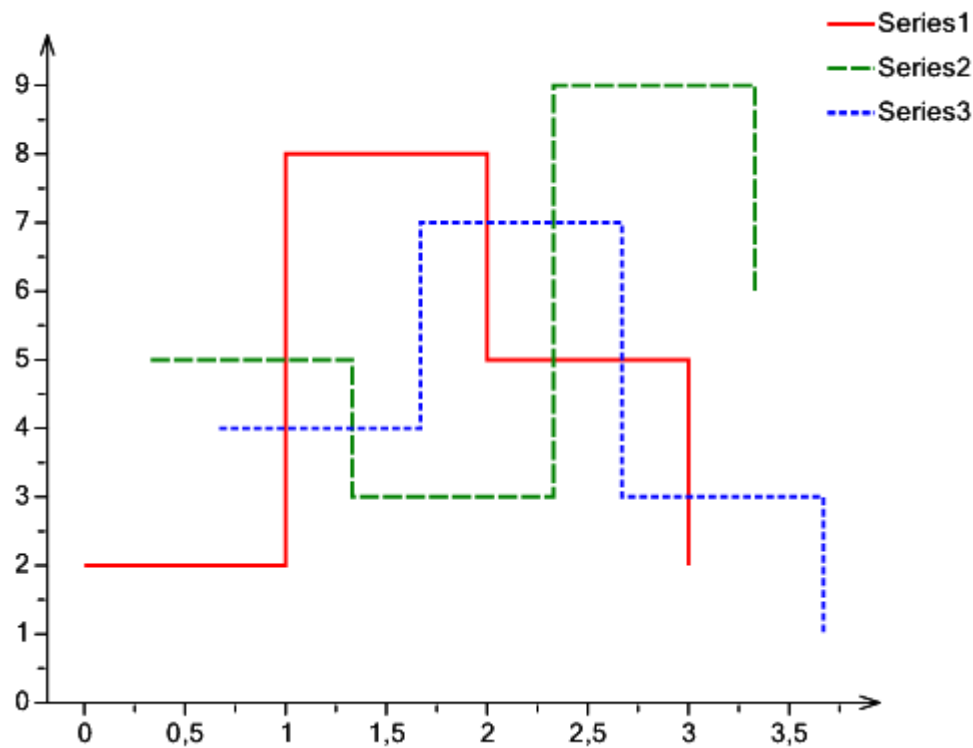
### Line Chart

Line chart is represented by the LineSeriesView, StepLineSeriesView, SplineSeriesView types.

The LineSeriesView creates broken line, passing through all series points.

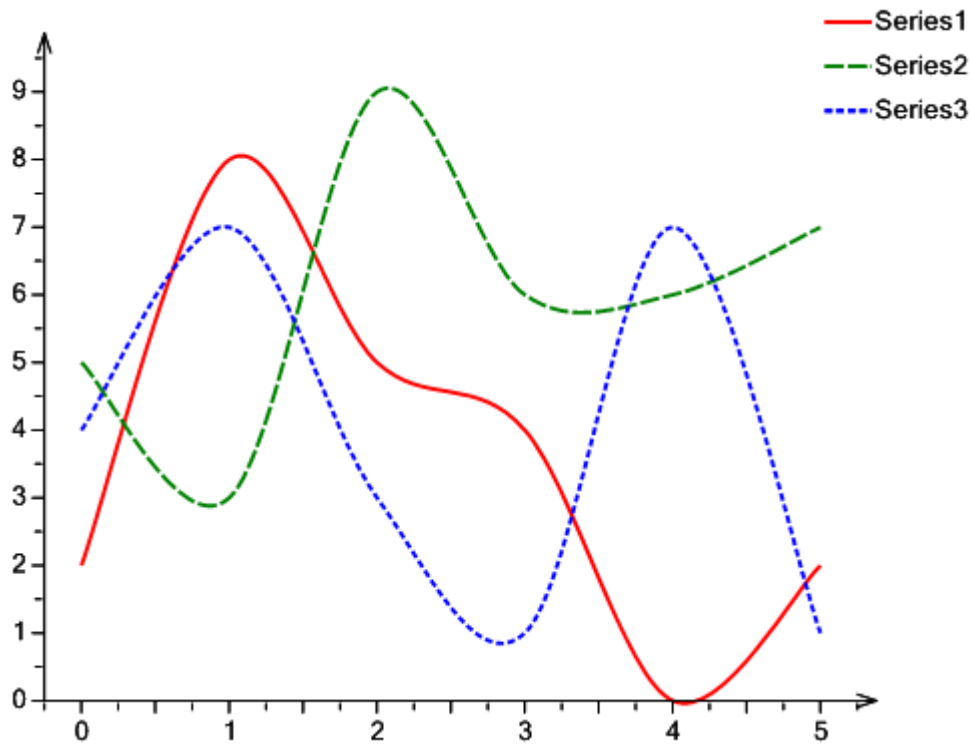


The StepLineSeriesView visualizes data as a step line, consisting of vertical and horizontal intervals and passing through all series points.



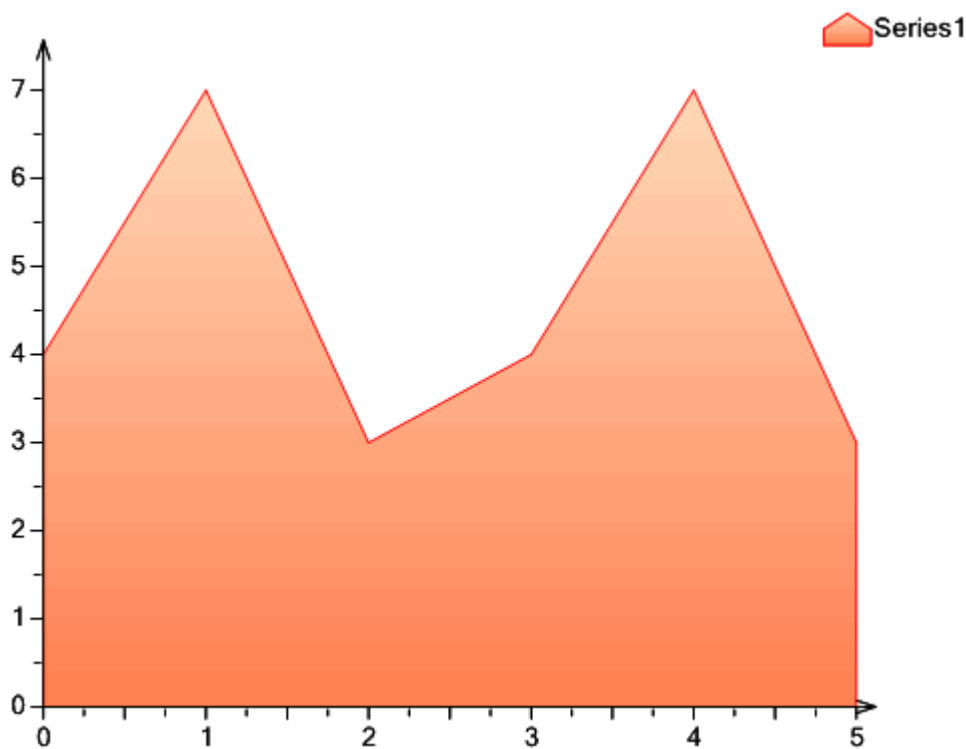
The SplineSeriesView represents data as a spline, passing through all series points.



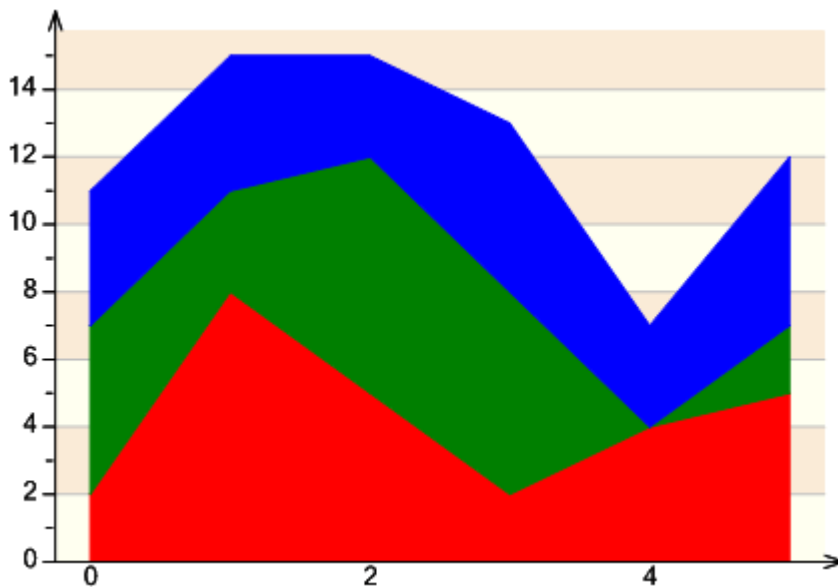


### Area Chart

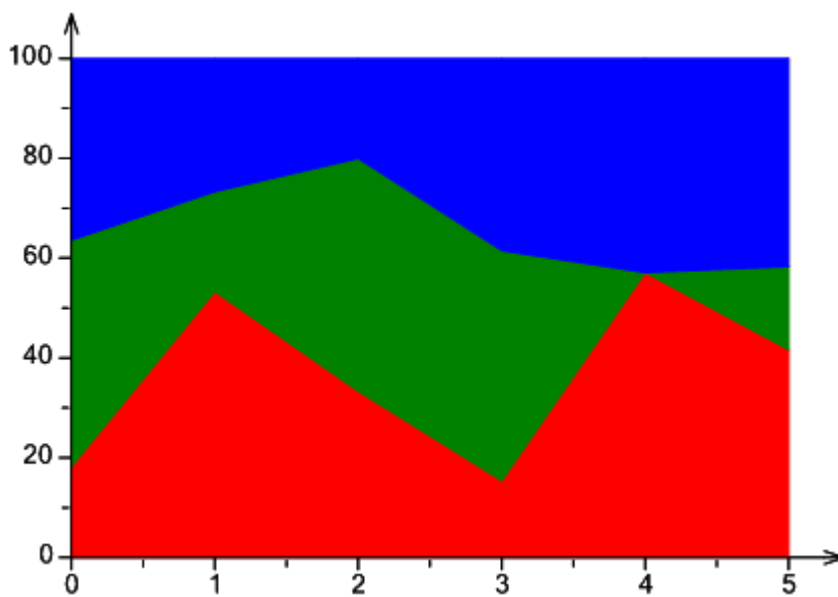
The AreaSeriesView represents data as area, bounded by a broken line, passing through all series points.



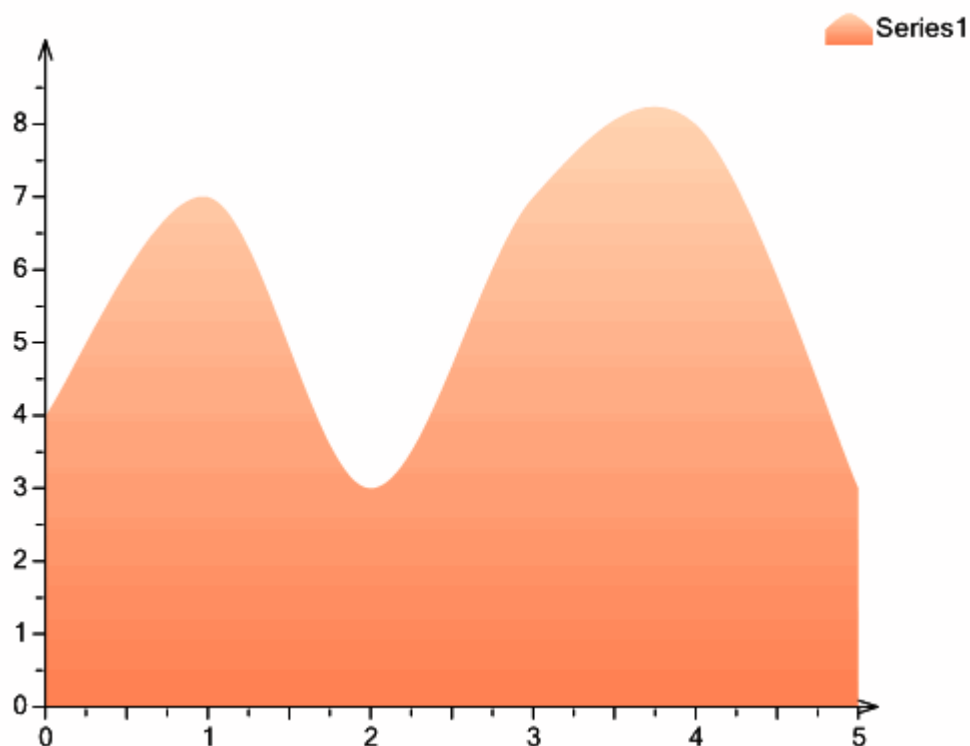
The StackedAreaSeriesView is a chart, containing areas, corresponding to different series. Series are located one over another.



The Stacked100AreaSeriesView is a chart that contains areas corresponding to different series. Series are located one over another, but they demonstrate relative value from total sum of series values at the given point.

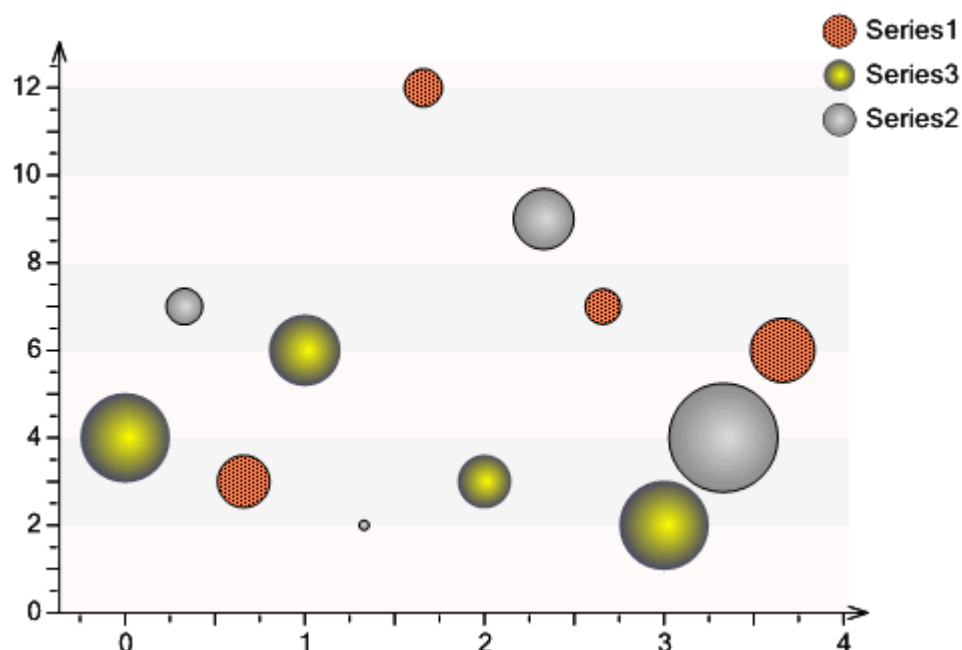


The SplineAreaSeriesView represents data as an area, bounded by spline, passing through all series points.



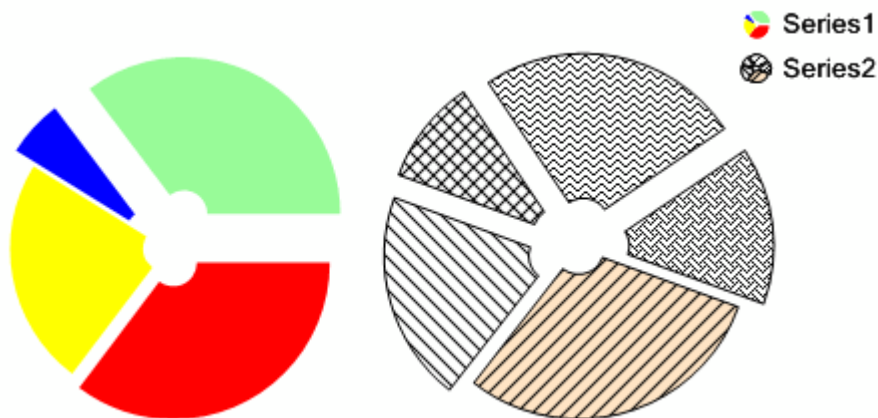
### Point Chart

The PointSeriesView displays data as circles with different diameters. A circle with Radius and Center at the point, assigned by coordinates, which are set by Argument and Value, is created for every series value. The MinRadius and MaxRadius properties set minimal and maximal radius of the displayed circles.

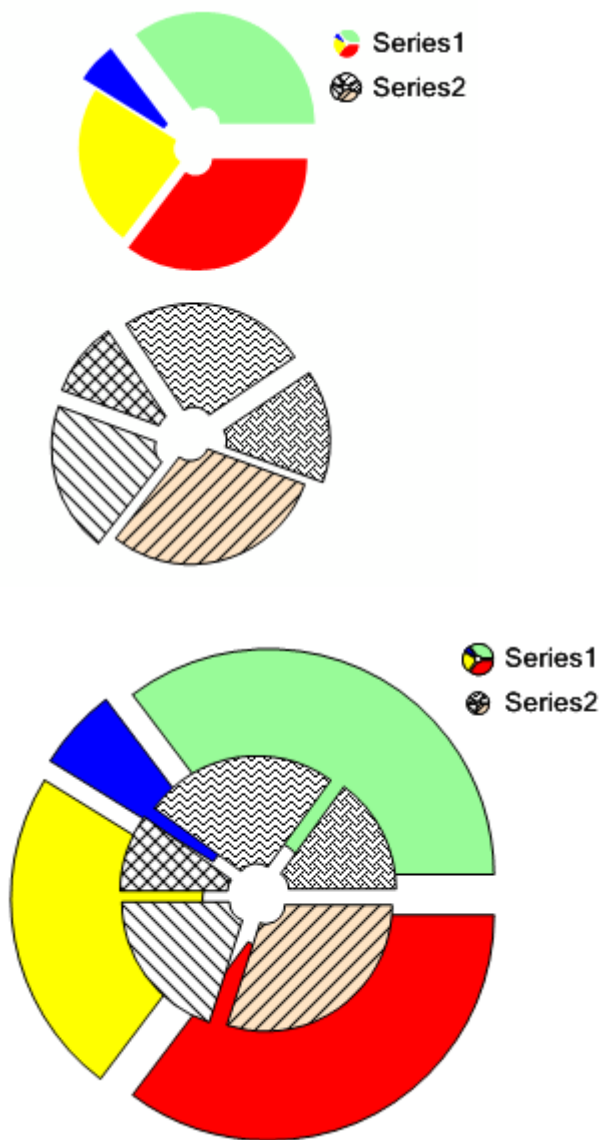


### Pie Chart

The PieSeriesView represents data as Pie. Every value is displayed as a separate segment, which angle depends on a ratio of the given value to the total amount of the whole series. You can assign fill for every segment, its shift from center, and internal and external radii.



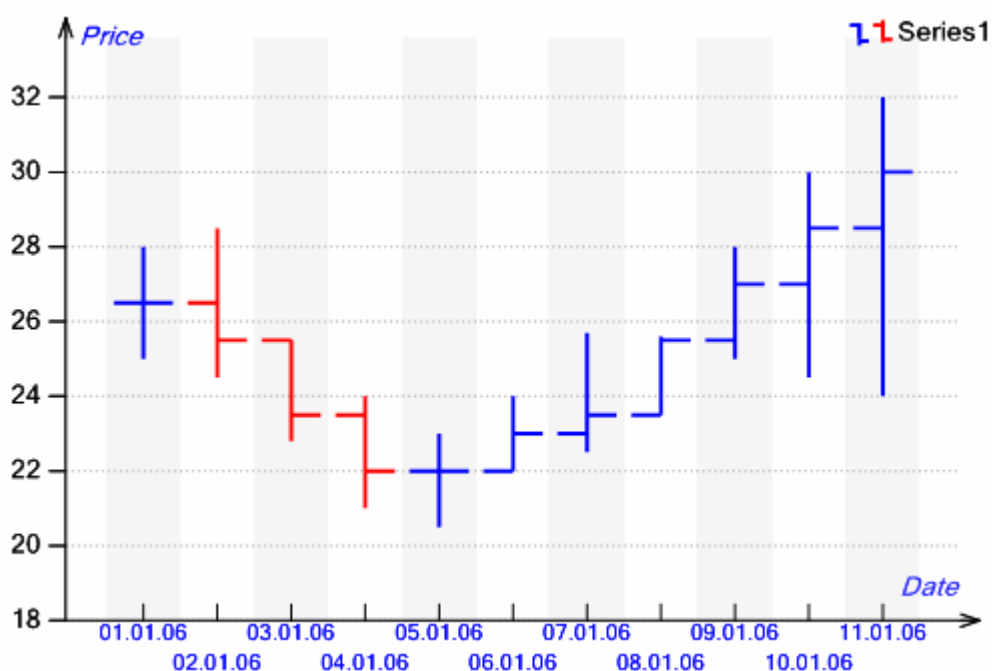
If the ChartArea contains multiple series of the PieSeriesView type, they can be located in the following ways: horizontally, vertically, and one inside another.



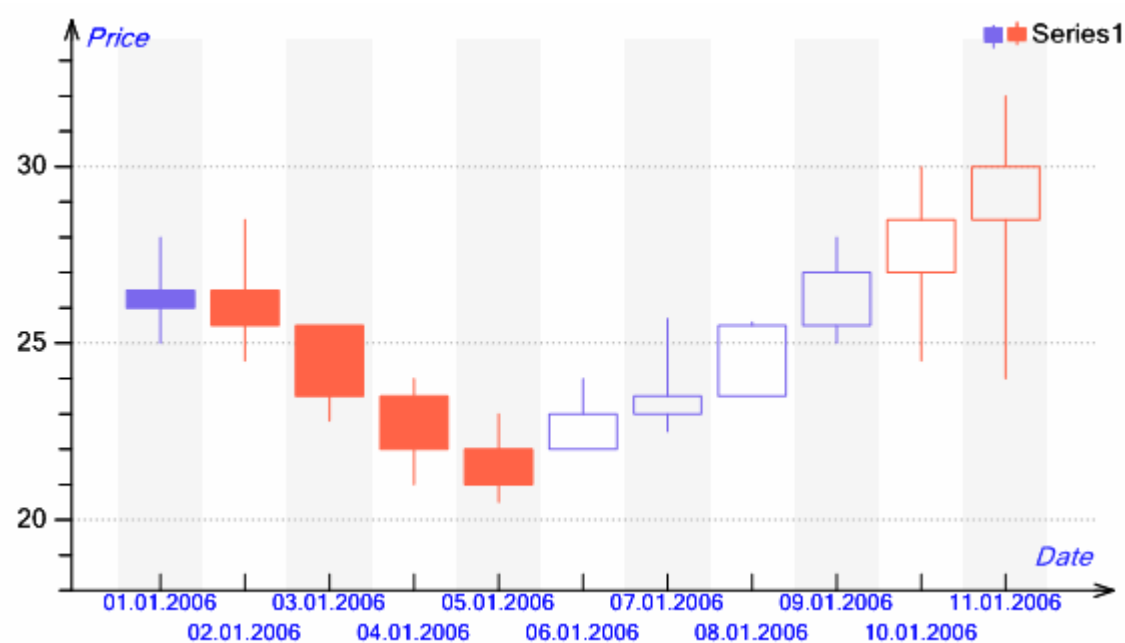
## Stock Chart

The Stock chart is represented by the StockSeriesView and CandleStickSeriesView. The StockSeriesView is used to display stock information with parameters Hi, Lo, Open, Close. Size and position of the line is defined by Hi and Lo properties. Rectangle size is defined by Open and Close properties.

Color of the rectangle line and fill is set by the Color and ReductionColor properties. If the current field, assigned by the ReductionValue, is less than the corresponding field of the previous series value, ReductionColor will be used, otherwise Color will be used. Rectangle is filled, if the Open value is less than the Close value.




The CandleStickSeriesView is used for displaying stock information with parameters Hi, Lo, Open, Close. Lines color is defined by the Color and ReductionColor properties.



SharpShooter Charts allows combining series of different types in a single chart.



### Series and Assignment of a Data Source

The Series element (  ) is intended for storing and displaying data. Chart type is assigned by the View property. Value Axis and Argument Axis are set by the ArgumentAxis and ValueAxis properties. The ScaleType property defines type of the argument scale and takes Qualitative and Numerical values. If ScaleType is set to Numerical, all series arguments are set in double type and sorted. In order to assign arguments, that are not set in double type (for example, string or date), it is necessary to set the ScaleType to Qualitative.

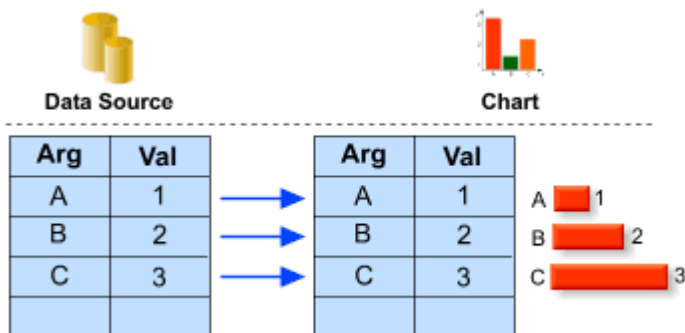
These series can be assigned with the use of a data source or by filling the Values collection. Independently of the way data is received, it represents a collection of objects of the SeriesValueBase type. When the chart is built, each data source value becomes corresponding to a single chart value. In common case, the SeriesValeBase contains the Argument of the object type and the UserData property of the object type, intended for storing secondary data. The SeriesValueBase descendants contain additional fields, required for building a specific chart. In simple case, the SeriesValue object also contains the Value property of the double type.

### Data Binding

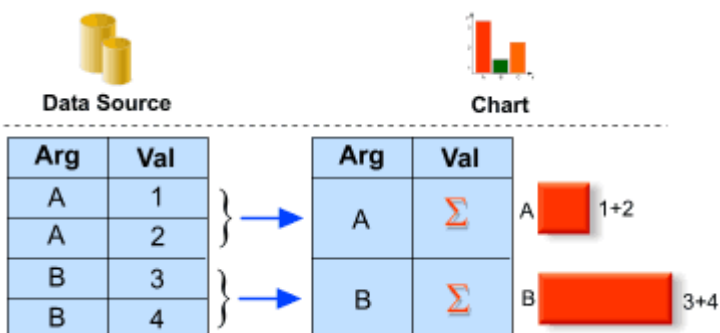
The SharpShooter Charts allows using data, received from ADO.NET objects or from various collections, as a data source for building charts. Any object, realizing the IList interface, can be used as a data source. Data source is assigned by the Series DataSource property.

Data binding is executed by making match between object fields of the SeriesValueBase type and the data source fields. Every match is assigned by the SeriesBindingType object, containing the SeriesValueName and PropertyName string properties, which specify name of the SeriesValueName property and name of the property (or the expression) from a data source correspondingly. The SeriesBinding objects are stored in the Bindings property of the SeriesBindingCollection type. Thus, the Bindings property describes match between data source fields and series value fields.

If the series AutoArgument property is set to true, the Argument field value will be assigned automatically in the ascending order, and it is not obligatory to set match for the Argument field. In simple case, when a data source and a collection of matches are assigned, only one series will be built. Such data binding is depicted schematically in the image below.

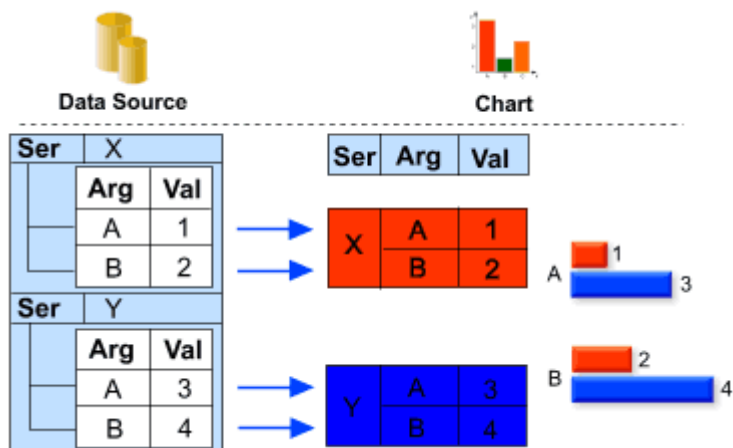


When a data source contains two or more similar values, these values will be summarized when building a chart.

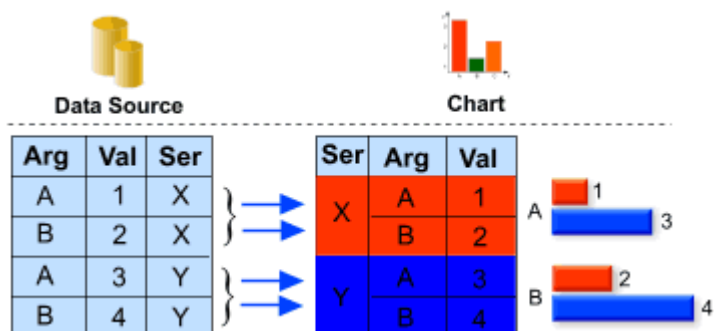


If some data source field is an IList list, it is possible to build series by data from nested lists, using the DataMember property. It is necessary to specify name of the field, containing all nested data, in the DataMember property. When you assign matches, you can use names of the properties of this nested data source.

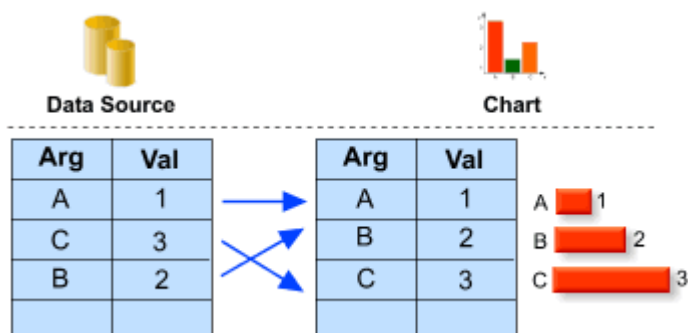
Building chart by nested data sources is executed in the following way: a separate series is created for every object from a data source. The Series values are got from the corresponding nested data source. This can be schematically showed in the following way:



Sometimes it is necessary to build multiple series by a single data source, where data is grouped in separate series by some parameter. You can do it, using the SeriesSplitter property. In the property, an expression, which result will be used to split data to series, is assigned.



Data inside every built series can be sorted. Sorting is assigned in the Sort property, and type of sorting (ascending or descending) is assigned by the SortingType property.



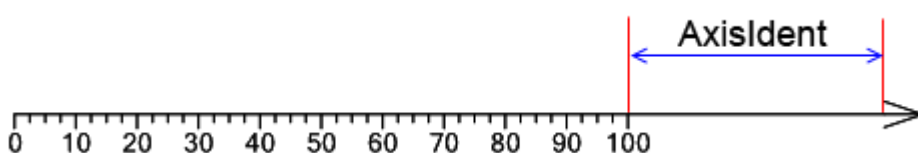
### Axis and Axis Elements

The required condition for designing most chart types is the availability of two perpendicular axes, in line of which arguments and chart point values will be laid. Binding series to axes is executed by the ArgumentAxis and ValueAxis properties. Axis type doesn't depend on its direction. But Argument axis is a horizontal axis, and Value axis is a vertical axis by default.

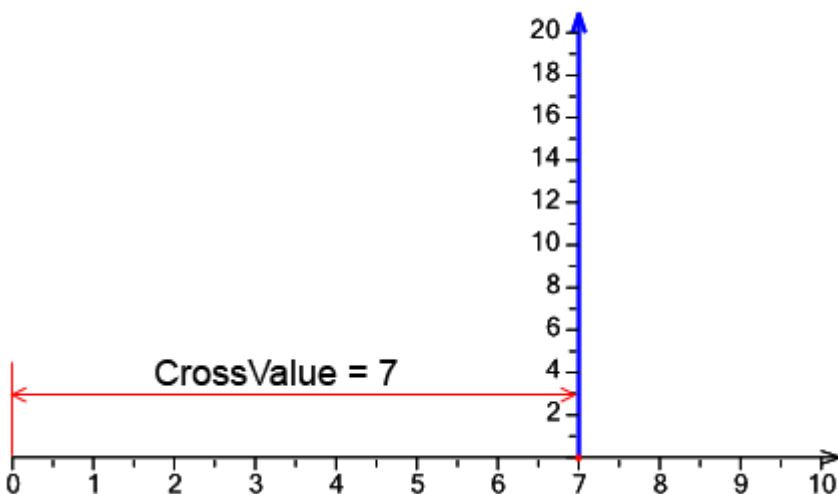
Axis is a scale, in line of which chart values are laid. Scale range is assigned by the MinValue and MaxValue properties, defining minimal and maximal scale values. Axis can automatically calculate scale range depending on taken series point's values, using this axis. It is necessary to set the MinValue and MaxValue properties to Auto. If the MinValue and MaxValue take constant values, scale range is assigned by these values regardless of automatically calculated values. The MinValue and MaxValue can take relative value in percentage from the automatically calculated range. If automatic range calculation is used, range can be completely negative or completely positive. The StartFromZero property defines if the range will include the 0 point.

The IsDiscrete property specifies if axis will display discrete or numeric values. This property allows using axis as argument axis for series, containing nonnumeric values.


Axis is a vertical or horizontal line that can be stretched to fit the ChartArea height or width subject to sifts, assigned by the Margins property. Axis appearance is set by the Stroke and Arrow properties. Stroke sets type, color, and width of the line. Arrow assigns axis arrow type. Axis direction is set by the Direction and Reverse properties. The AxisIndent property sets distance between axis arrow and end point of the scale.



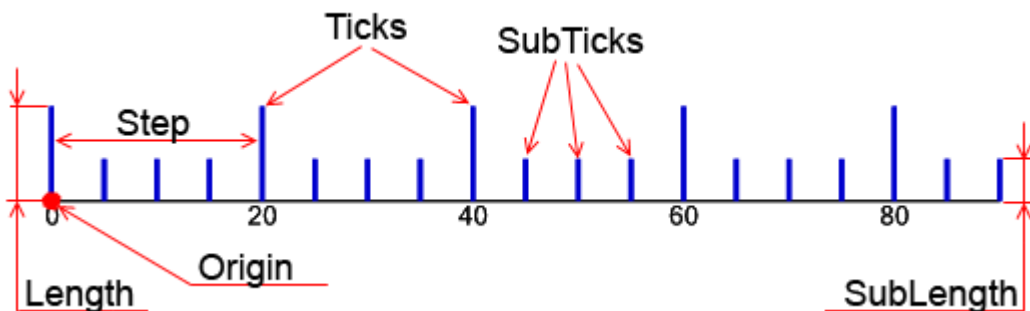
The CrossValue property assigns axis position relatively to the other axis, set by the PerpendicularAxis property.




The SharpShooter Charts contains elements intended for displaying ticks, labels and other information on the scale.


The Ticks element  is designed for displaying ticks on the scale. Ticks width is set by the Stroke property. Divisions step is assigned by the Step property. If Step is Auto, number of divisions is assigned in the Divisions property. The Origin property sets a bench mark for divisions. If the UseRoundValues property is set to true, scale ticks will be distributed in the way that the corresponding values contain minimal amount of the fractional numbers; the number of divisions will be less or equal to the Divisions property value. Ticks length is set in the Length property. In the range between ticks, sub-ticks are displayed. They divide this range in accordance with the SubDivisions property. Sub-ticks length is set in the SubLength property.

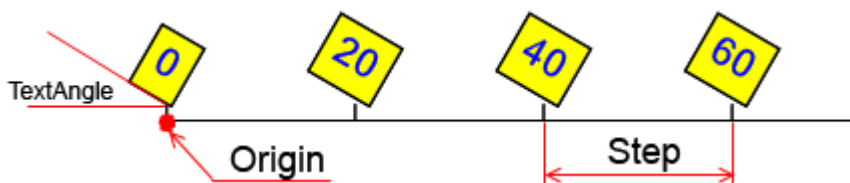






Ticks and sub-ticks relative position can be assigned by the `SubTicksPosition` property. When the element is drawn, ticks will be evenly allocated along the scale. The `UseDiscreteValue` property is used for displaying ticks on the discrete scale. If this property is set to true, ticks will be displayed only for discrete scale values.

The `ScaleMarks` element  is intended for displaying scale marks. Mark type is set by the `Shape` property, mark size is defined by the `MarkSize` property. Sub-marks size is defined by the `SubMarkSize` property. Logic and properties, assigning marks position on the scale, are analogous to those of the `Ticks` element.

The `ScaleLabels` element  is designed for displaying axis values. If the axis is discrete, `ScaleLabels` will display discrete axis values. If the axis is non-discrete, labels will be displayed evenly along the scale. Divisions step is set by the `Step` property. If step is set to auto, number of divisions is assigned in the `Divisions` property. The `Origin` property assigns bench mark for divisions. If the `UseRoundValues` property is set to true, scale labels will be distributed in the way that the corresponding values contain minimal amount of the fractional numbers; the number of divisions will be less or equal to the `Divisions` property value. Label angle is set in the `Angle` property, Label orientation is set by the `TextRotationMode`. The `Fill` and `Stroke` properties set fill type and bounds of the area, occupied by the label. This area can be enlarged by the `ItemMargins` property. The `ShowSuperposableLabels` specifies if the superposable scale labels will be displayed.



The `CustomLabels` element  is similar to the `ScaleLabels` element, but it allows assigning arbitrary scale marks, not bound to axis values. Every label is assigned by the couple "text" – "scale value" and is stored in the `Labels` collection. If one or multiple labels contain the `Auto` value, such labels are evenly allocated on the scale.

The `ScaleTitle` element  is designed for assigning axis title. Label, set by the `Text` property, is located at the end of the axis.

The `ScaleLabels`, `CustomLabels`, and `ScaleTitle` elements feature several properties, assigning appearance and location of the labels. The `OddLabelsDistance` sets indent of odd labels relatively to even ones. The `TextAlingment` property assigns labels position in relation to the axis point, corresponding to the label value. The `Position` property sets labels position in relation to the highest label. The `Formula` property sets an expression for calculating label value, and the `TextFormal` property assigns string for label text formatting.

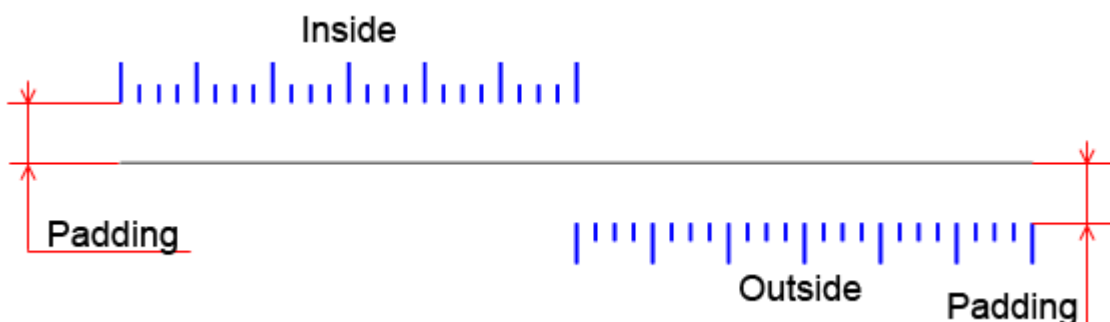
The `Ticks`, `ScaleMarks`, `ScaleLabels`, `CustomLabels`, and `ScaleTitle` elements feature a set of similar properties.

If there is the need to display axis elements only within the specified range, you can use the MaxLimit and MinLimit properties assigning range bounds. The UseLimits property specifies that limitation on elements display will be used.

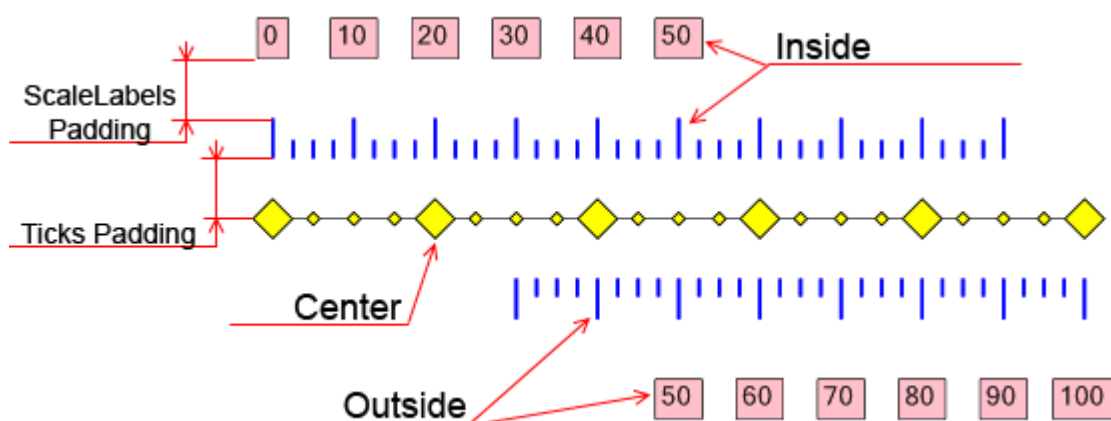
Elements shift from the scale is set by the Distance property. Often axis contains multiple scale elements, for example, Ticks and ScaleLabels. The Dock property assigns relative position of such elements. If the element's Dock property is set to None, element's position on the axis doesn't depend on any other axis elements. In this case element's position is defined by the Distance property value.



If the Dock property is set to Center, an element is positioned at the axis center, regardless any other axis elements. If the Dock property value is Inside, an element is located inside the axis at the distance, assigned by the Padding property. If the Dock property value is set to Outside, an element is positioned outside the axis at the distance, defined by the Padding property.



If the axis contains multiple elements, which Dock property value is not None, at first, elements with the Dock property value, set to Center, are positioned, and then all other elements are allocated on the axis in the order, they were added to the axis, subject to the size of already positioned elements.

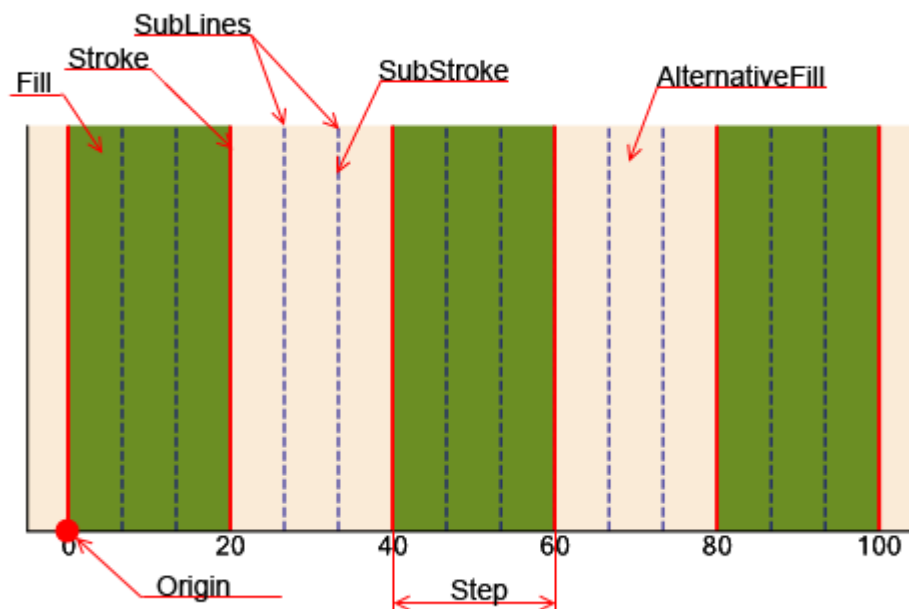


The Colorizer property allows setting scale elements' color. It is also possible to set color of all axis elements by assigning the axis Colorizer property.

The GridLines element is designed for displaying values grid in the chart. Grid step is assigned by the Step property. If this property value is set to Auto, a number of divisions will be set by the Divisions property. The Origin property defines a bench mark. If the UseRoundValues property is


set to true, scale labels will be distributed in the way that the corresponding values contain minimal amount of the fractional numbers; the number of divisions will be less or equal to the Divisions property value. Amount of sub-lines is set in the SubLinesCount property.

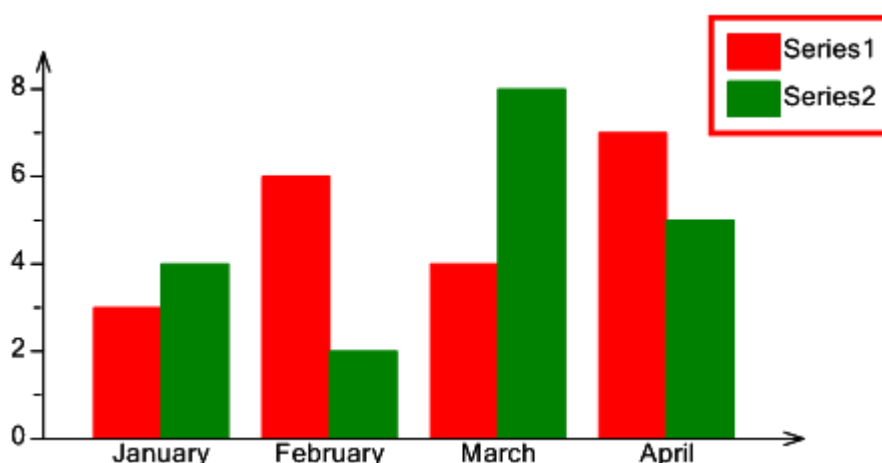
The Fill and AlternativeFill properties assign fill for alternate lines. The Stroke and SubStroke properties define type of Lines and Sub-lines.



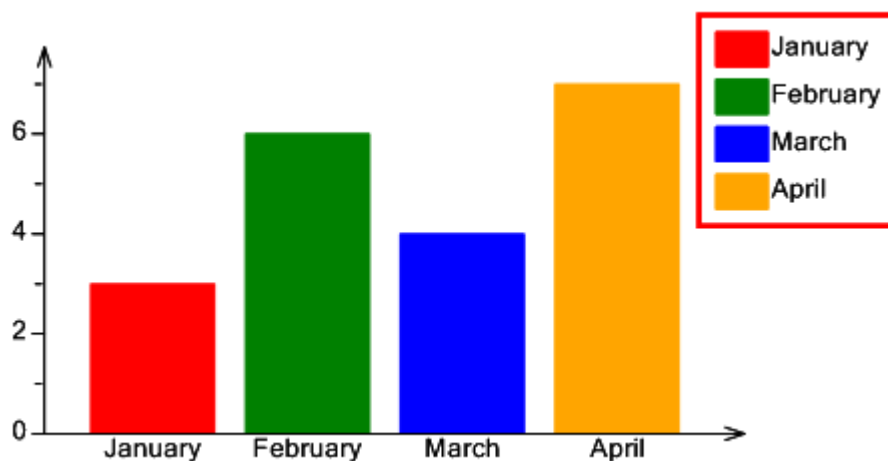
### Additional Chart Elements

The SharpShooter Charts contains elements, designed for facilitating chart data comprehension. They also fulfill some secondary functions.

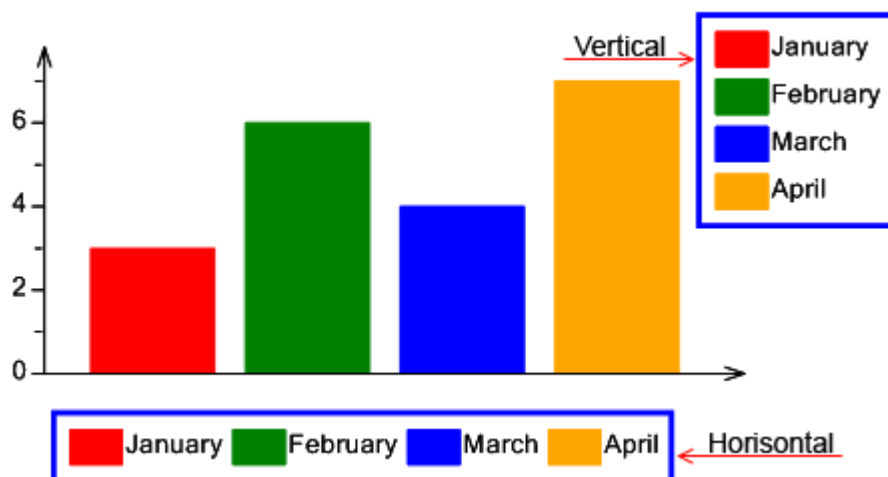
The Legend element  is designed for displaying conventions used in chart. The series DisplayingInLegendType property assigns type of displaying information about series in the Legend. This property can take None value, when series information is not displayed, and Series value, when series information is displayed.



If the DisplayingInLegendType property is set to Values, information on every series value will be displayed.

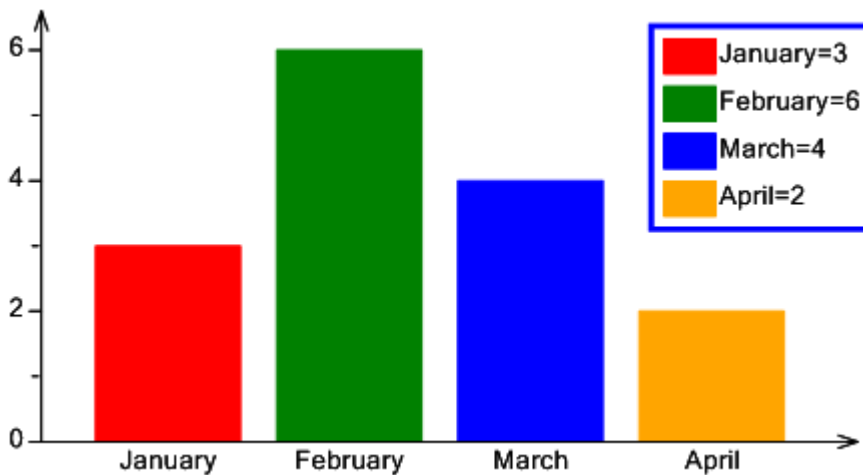


Legend's center position is assigned by the Center property. Legend items can be located both vertically and horizontally. It is set by the LegendType property.

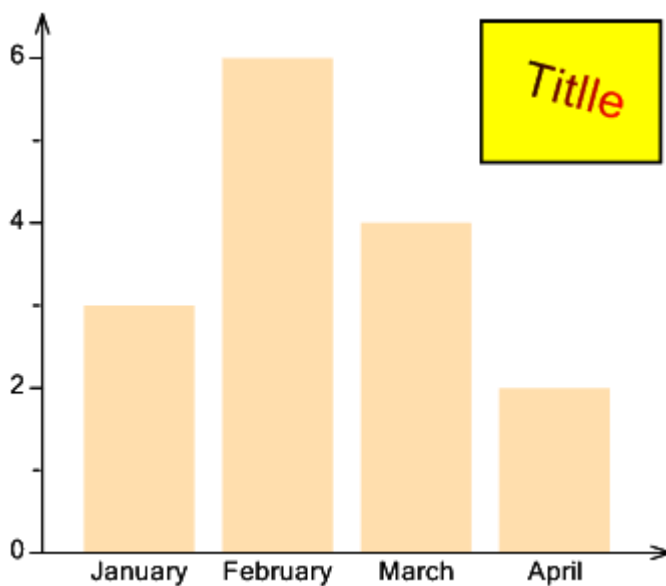


The ColumnsCount property defines the number of columns in the Legend. The ItemMargins and Margins properties assign shift of every legend item and of the whole legend correspondingly. The Padding property assigns shift from the legend boundaries.

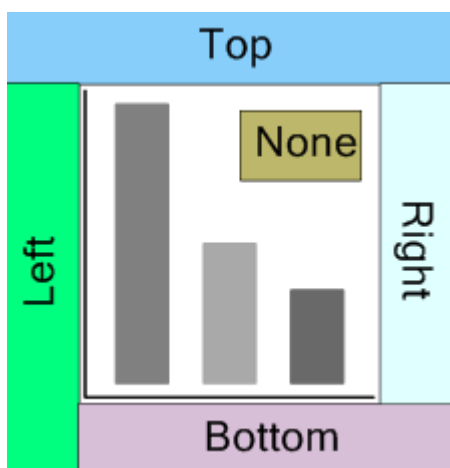
The LegendExpression property sets expression, which result will be displayed in the Legend. Expression can be set for displaying information on series as well as for displaying information on every series value. For example, if you set the following expression "Argument & '=' & Value", you will get the Legend as represented in the image below.



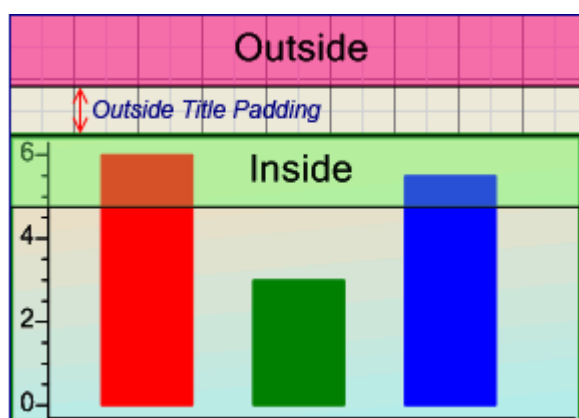
The Title element **T** is designed for creating chart title and various labels, not bound to data. Label text is set in the Text property, text font is set in the Font property. The TextFill and Fill properties assign fill of Title text and background correspondingly. The stroke property defines Title boundaries. The Angle property assigns text rotation angle in degrees. The Margins property sets indents between Title text and Title boundaries. The Padding property sets shift of the Title bounds from the area, filled with color.



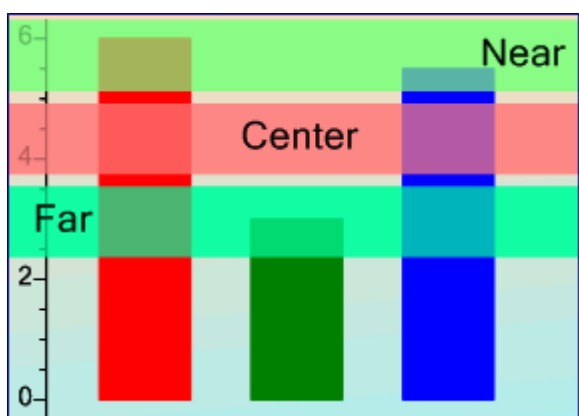
The Title and Legend elements can be automatically positioned by binding these elements to the ChartArea boundaries. The DockingType property defines side, element will be docked to. It can process the following values: None, Left, Right, Top, and Bottom. If the property value is set to None, binding to ChartArea sides is not performed, and element can be located in the arbitrary position.



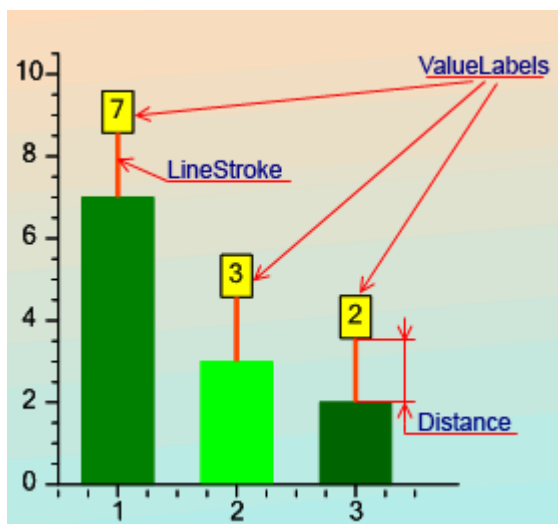
The DockingStyle property defines position, element will be located at: inside or outside the ChartArea. If the DockingStyle property is set to Outside, element will be located outside the ChartArea, and the area, dedicated for building the chart, will be decreased. If the DockingStyle property is set to Inside, element will be located inside the ChartArea, and it will partly overlay the area, dedicated for building the chart.



The Alignment property sets alignment of the element content inside the selected area.

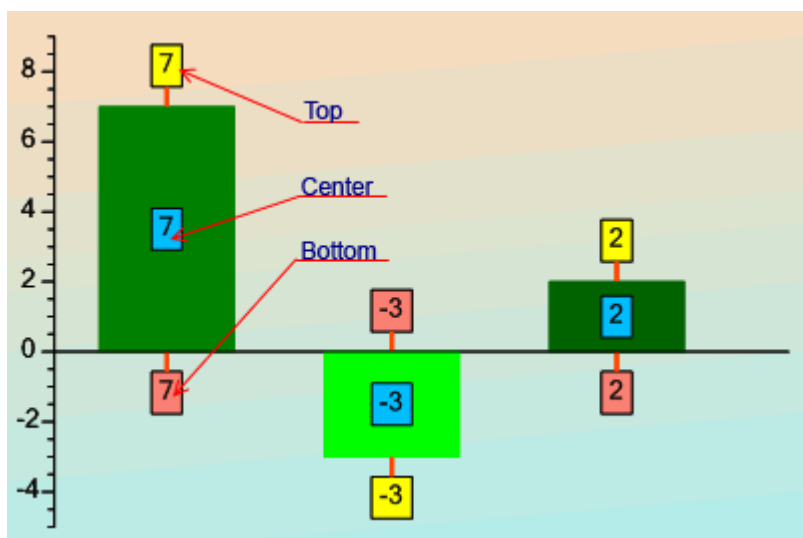


The ValueLabels element  is designed for displaying value or other information on every series point.




The Distance and Shift properties assign ValueLabel shift relatively to the point, corresponding to this value. ValueLabels angle is set in the Angle property. The LineStroke property defines type of the line, connecting label to a corresponding point.

The Position property defines label position in the chart and can process the following values: Top, Center, and Bottom.

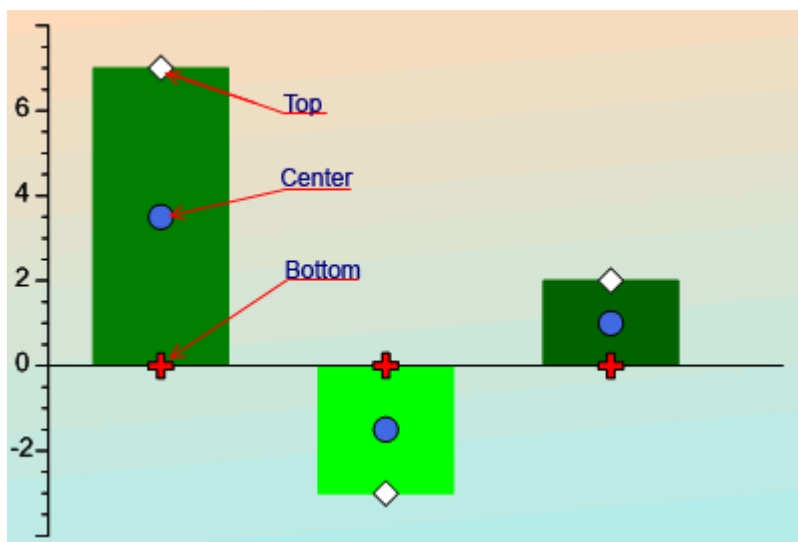


The Expression property allows you to assign an expression, which result will be displayed in the Label.

The ValueMarkers element  is designed for displaying graphical marks in series. Mark type is set by the Shape property, Mark size is set by the Size property.

The Distance and Shift properties assign element shift relatively to the point, corresponding to the value. The LineStroke property defines type of the line, connecting mark and a corresponding point.

The Position property defines mark position in the chart and it can process the following values: Top, Center, and Bottom.



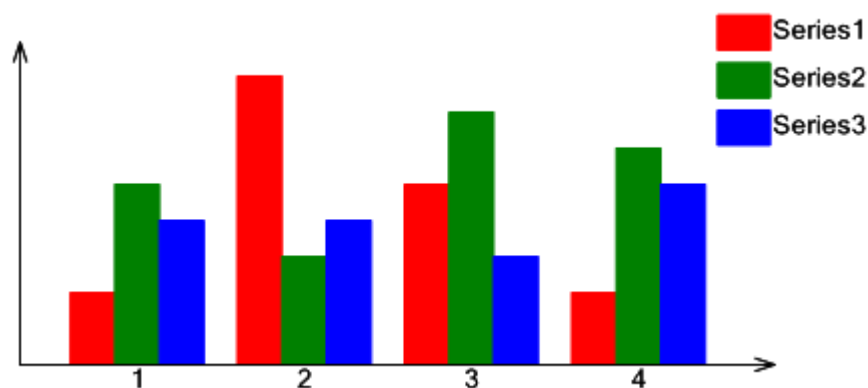
### Additional Features

#### Palettes

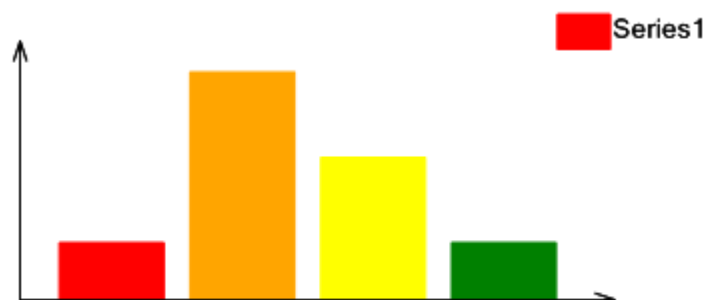
The SharpShooter Charts provides the ability to set color to series and separate values by using pre-defined sets of colors, line types, and figures.

Palettes assign color and fill to series or separate series elements. They also assign color and type of line for linear charts, marks type for the ValueMarkers element.

The ChartArea Palette property sets series palettes, and Series Palette property sets palette for series elements. The image below demonstrates the use of palette in the ChartArea. The ChartArea contains three series, and series Fill property is not set. In this case, series use palette colors.



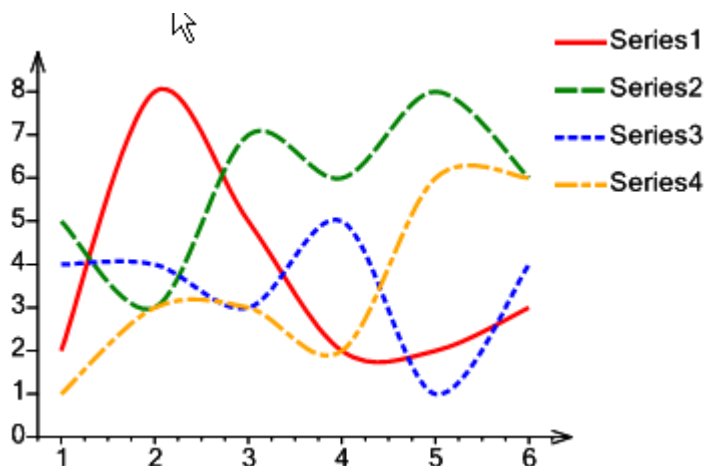
The series Palette property defines color for every series element.



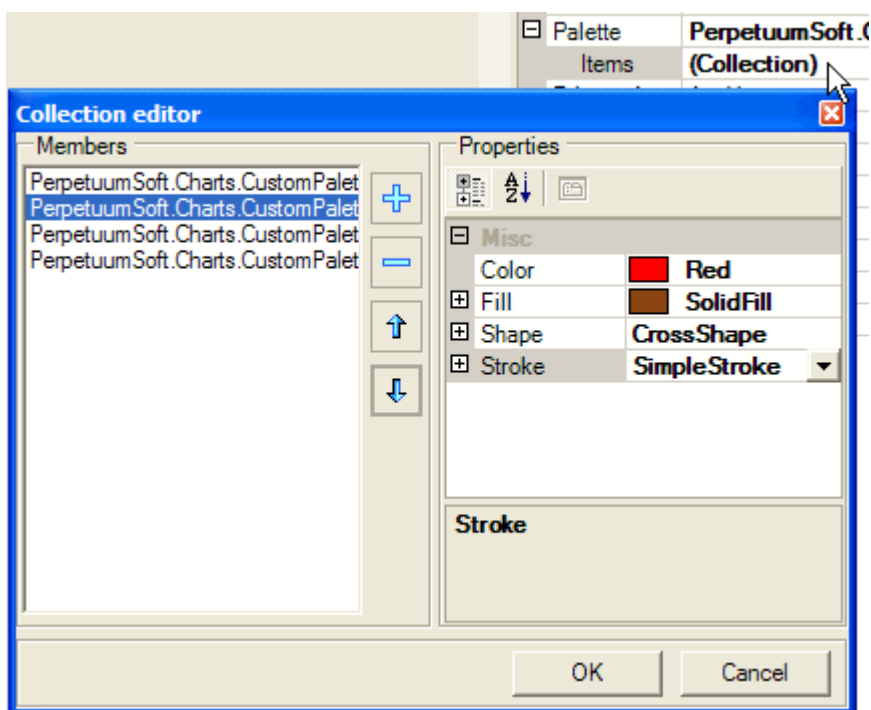





Palette property of the linear charts sets not only line color, but also line type.



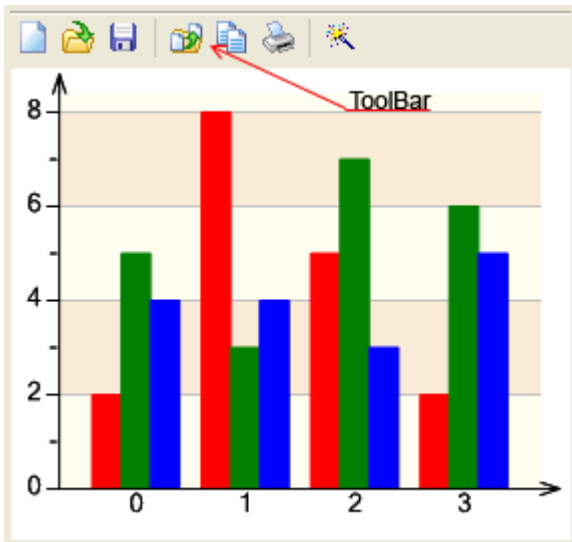
The CustomPalette class provides the ability to set custom palettes. It consists of the set of objects; each of them contains fill, color, shape type for displaying ValueMarkers and line type in the element




## The Chart Viewer


ChartViewer  is a component, designed for displaying charts in .NET Windows Forms applications. The Chart object can be assigned by the Chart property as well as by the ChartStream property. It is possible to upload the chart, serialized in the string, in the ChartViewer.


A ToolBar is situated at the top of the ChartViewer, the remainder is intended for displaying the chart.





The ToolBar contains the following buttons:


The "New Chart"  toolbar button creates a blank chart;

The "Open Chart"  toolbar button calls dialog for opening the chart, saved to a file;

The "Save Chart"  toolbar button calls dialog for saving the chart to a file;

The "Save As Image"  toolbar button calls dialog for saving the chart as an image;

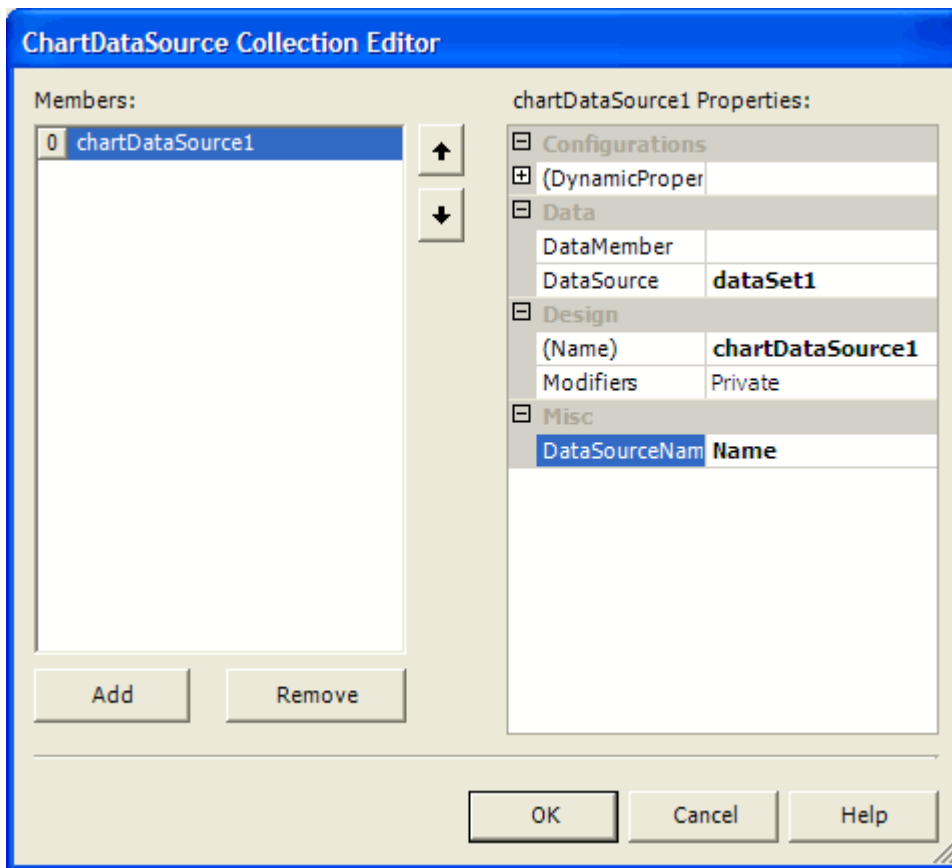
The "Copy to clipboard"  toolbar button copies the chart as an image to clipboard;

The "Print"  toolbar button calls print dialog;

The "Run designer"  toolbar button runs designer.

Buttons visibility on the toolbar is set by the ShowNew, ShowOpen, ShowSave, ShowSaveAsImage, ShowCopyToClipboard, ShowPrint, ShowDesigner properties, and toolbar visibility is set by the ToolBarVisible property.

The DataSource property assigns data sources for the chart.

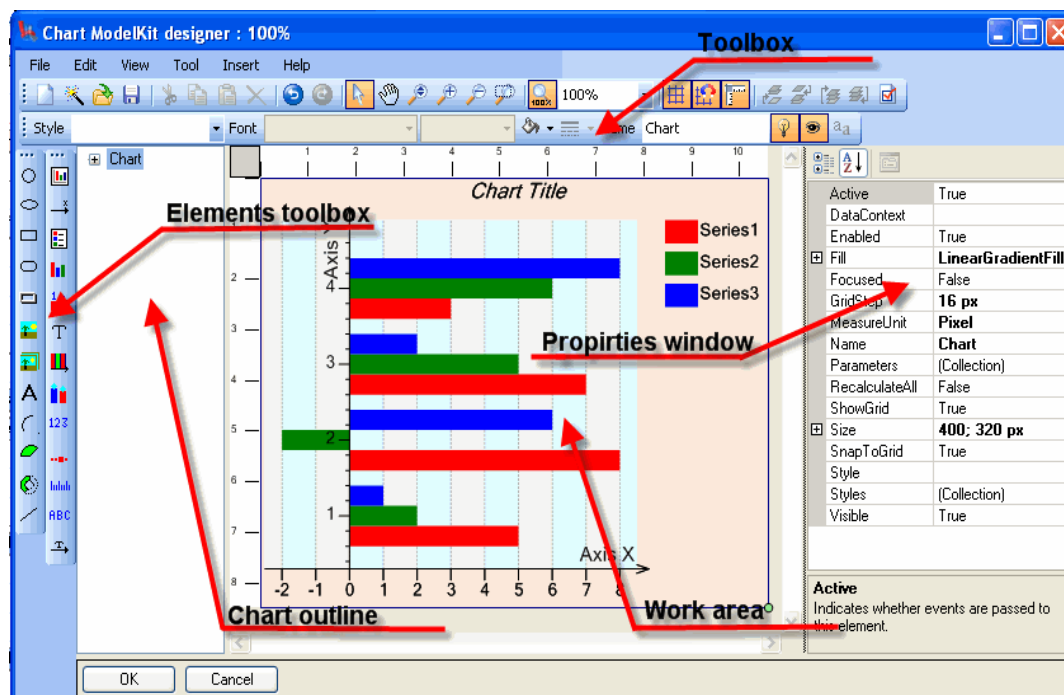


In further development, while assigning series data source, it is necessary to specify the DataSourceName.

At run time, designer is called by double click on the ChartViewer.


## The Designer

The SharpShooter Charts Designer is used for creating charts. Designer appearance is shown in the image below. Chart is an object, containing various elements. Every element has a set of properties which are displayed in the Properties Window. You can edit properties in this window.




Visual and non-visual elements are available through the Insert menu or buttons on the ToolBox. In order to place an element into the chart, you should select this element in the Insert menu or on the Toolbox and then click the mouse button in the work area or holding the mouse button down, stretch the element to the required size. Chart structure is displayed in the Chart outline window. You can change elements position by dragging them with a mouse.

Let's consider functions available via the File menu. The commands for working with a file are grouped here.

File\New\Blank menu item (the  toolbar button) allows creation of a new empty chart.

File\New\Wizard menu item (the  toolbar button) allows creation of a new chart with the help of Wizard.

File\Open Ctrl+O menu item (the  toolbar button) – calls the dialog box for the chart loading from the file.

File\Open and merge menu item calls the dialog box for loading the chart from a file. At the same time a new chart is added to the existing one.

File\Open in wizard and merge menu item calls the Wizard loading the chart. At the same time a new chart is added to the existing one.

File\Save Ctrl+S menu item (the  toolbar button) saves the chart.


File\Save As menu item is used for saving the chart with a new name.


File\Export menu item exports the chart into one of popular bitmapped or vector graphics formats (JPEG, GIF, PNG, BMP, SVG and Macromedia Flash) and calls the save dialog box.





You can view the recent files history including file system paths below the separator.


Let's consider the functions available via Edit menu. The commands for editing chart are grouped here.


Edit\Undo Ctrl+Z menu item (the  toolbar button) means the cancel of the changes made in the chart.

Edit\Redo Ctrl+Y menu item (the  toolbar button) means the repetition of the canceled actions when editing the chart.


Edit\Cut Ctrl+X menu item (the  toolbar button) means the copying and deleting the selected elements.


Edit\Copy Ctrl+C menu item (the  toolbar button) means the copying of the selected elements.


Edit\Paste Ctrl+V menu item (the  toolbar button) means the pasting of elements from the buffer to the selected element or its parent, if the selected element is not a container.

Edit\Delete Ctrl+Del menu item (the  toolbar button) is intended for deleting selected elements.


Let's consider functions available via the View menu. The commands for the designer appearance control are grouped here.


View\Show rulers menu item (the  toolbar button) is intended for switching on/off the rulers display in the Working area (Work area)


View\Show grid menu item (the  toolbar button) is intended for switching on/off the grid display in the Working area (Work area).


View\Snap to grid menu item (the  toolbar button) is intended for switching on/off the snap of elements' control points to the grid.


Let's consider functions available via the Tool menu. The commands of different tools switching on are grouped here.


The Tool\Select F2 menu item (the  toolbar button) is used to switch on the Select tool. The Select tool allows selecting elements in the Work area and editing elements via their designers.

Tool\Pan F4 menu item (the  toolbar button) is used to panning the picture of the working area (Work area).

Tool\Zoom in menu item (the  toolbar button) is used for zooming an element in.

Tool\Zoom out menu item (the  toolbar button) is used to zoom an element out.


Tool\Region zoom menu item (the  toolbar button) is used to zoom a selected region in.


Tool\Dynamic zoom menu item (the  toolbar button) switches on the Dynamic zoom tool to change zoom of the selected region.





Let's consider functions available via the Insert menu. The commands for adding different elements to the chart are grouped here. They are broken to groups depending on the element destination.

Elements, grouped in the Insert\Structure menu item, are intended for the chart structure assigning.


Insert\Chart\ChartArea menu item (the  button on the Elements toolbox) adds the ChartArea element.


Insert\Chart\Axis menu item (the  button on the Elements toolbox) adds the Axis element (the Axis element).


Insert\Chart\Legend menu item (the  button on the Elements toolbox) adds legend (the Legend element).

Insert\Chart\Series menu item (the  button on the Elements toolbox) adds chart series (the Series element).


Insert\Chart\ValueLabels menu item (the  button on the Elements toolbox) adds ValueLabels (the ValueLabels element).


Insert\Chart\Title menu item (the  button on the Elements toolbox) adds chart Title (the Title element).

Insert\Chart\GridLines menu item (the  button on the Elements toolbox) adds grid to chart (the GridLines element).


Insert\Chart\ValueMarkers menu item (the  button on the Elements toolbox) adds Value Markers (the ValueMarkers element).


Insert\Scale Elements menu item contains commands for adding element intended for scale design.

Insert\Scale Elements\ScaleLabels menu item (the  button on the Elements toolbox) adds element, intended for displaying values on the scale (the ScaleLabel element).

Insert\Scale Elements\ScaleMarks menu item (the  button on the Elements toolbox) adds element, intended for displaying scale marks (the ScaleMarks element).


Insert\Scale Elements\Ticks menu item (the  button on the Element toolbox) adds element, intended for displaying ticks on the axis (the Ticks element).


Insert\Scale Elements\CustomLabels menu item (the  button on the Elements toolbox) adds element, designed for displaying text labels on the axis (the CustomLabels element).

Insert\Scale Elements\ScaleTitle menu item (the  button on the Elements toolbox) adds scale title (the ScaleTitle element).

Insert\Primitives menu item contains commands for adding basic primitives such as shapes.


Insert\Primitives menu item contains the commands of adding the basic primitives, such as geometrical figures and others.


Insert\Primitives\Line menu item (the  button on the Elements toolbox) adds a line (the Line element).

Insert\Primitives\Circle menu item (the  button on the Elements toolbox) adds a circle (the Circle element).




Insert\Primitives\Ellipse menu item (the  button on the Elements toolbox) adds an ellipse (the Ellipse element).


Insert\Primitives\Rectangle menu item (the  button on the Elements toolbox) adds a rectangle (the Rectangle element).


Insert\Primitives\RoundedRectangle menu item (the  button on the Elements toolbox) adds a rectangle with rounded angles. (the RoundedRectangle element)


Insert\Primitives\Frame menu item (the  button on the Elements toolbox) adds a rectangle with a bevel (the Frame element).

Insert\Primitives\Picture menu item (the  button on the Elements toolbox) adds a picture (the Picture element).

Insert\Primitives\PictureSet menu item (the  button on the Elements toolbox) adds a picture set (the PictureSet element).

Insert\Primitives\Label menu item (the  button on the Elements toolbox) adds text inscriptions (the Label element).


Insert\Primitives\Arc menu item (the  button on the Elements toolbox) adds an arc (the Arc element).


Insert\Primitives\Pie menu item (the  button on the Elements toolbox) adds a pie (the Pie element).

Insert\Primitives\RingSector menu item (the  button on the Elements toolbox) adds a ring sector (the RingSector element).


The Help menu item contains the About item, intended for displaying information on the software.

There are also the buttons on the Tool bar, intended for changing output order of the selected elements.

Move selected objects to forward  raises the selected elements up to one position.

Move selected objects to back  lowers the selected elements down to one position.

Send selected objects to forward  raises the selected elements at the very top.

Send selected objects to back  lowers the selected elements at the very bottom.

If the selected elements group order is changed, the relational order inside the group is kept.

Let's consider working with the Select tool in the work area.

In the Select tool mode any visual element can be selected by clicking on it with the left mouse button. It is possible to select a group of elements either by means of selecting each of the elements while holding the Ctrl key pressed or by means of "lassoing" the group of elements holding the left mouse button pressed.

Control points in the form of circles are displayed in the selected elements.

The selected elements are highlighted in the chart structure tree.

The editing operations applicable to the selected elements:

Copying to clipboard;

Pasting from the clipboard to the element;

Deleting;

Changing output order;

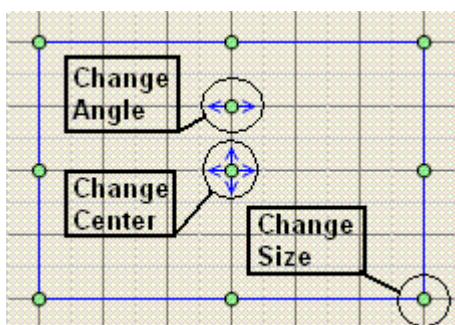
Changing properties in the Properties Window.

Changing element position with the help of a mouse.

Changing some element's properties by moving its control points.

Control points of the selected element are intended for its main properties changing. The point position can be changed by dragging it with the mouse.

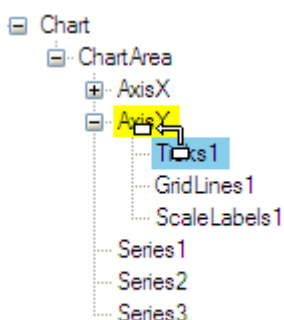
Each element has its own control points set. For example, rectangular elements have ten points, one of which is intended for changing center, another one is for changing rotation angle, and the rest of them are intended for changing size of the rectangular area.



The Control points for the rectangular element designer.

When control points position is changed, the corresponding element property changes as well.

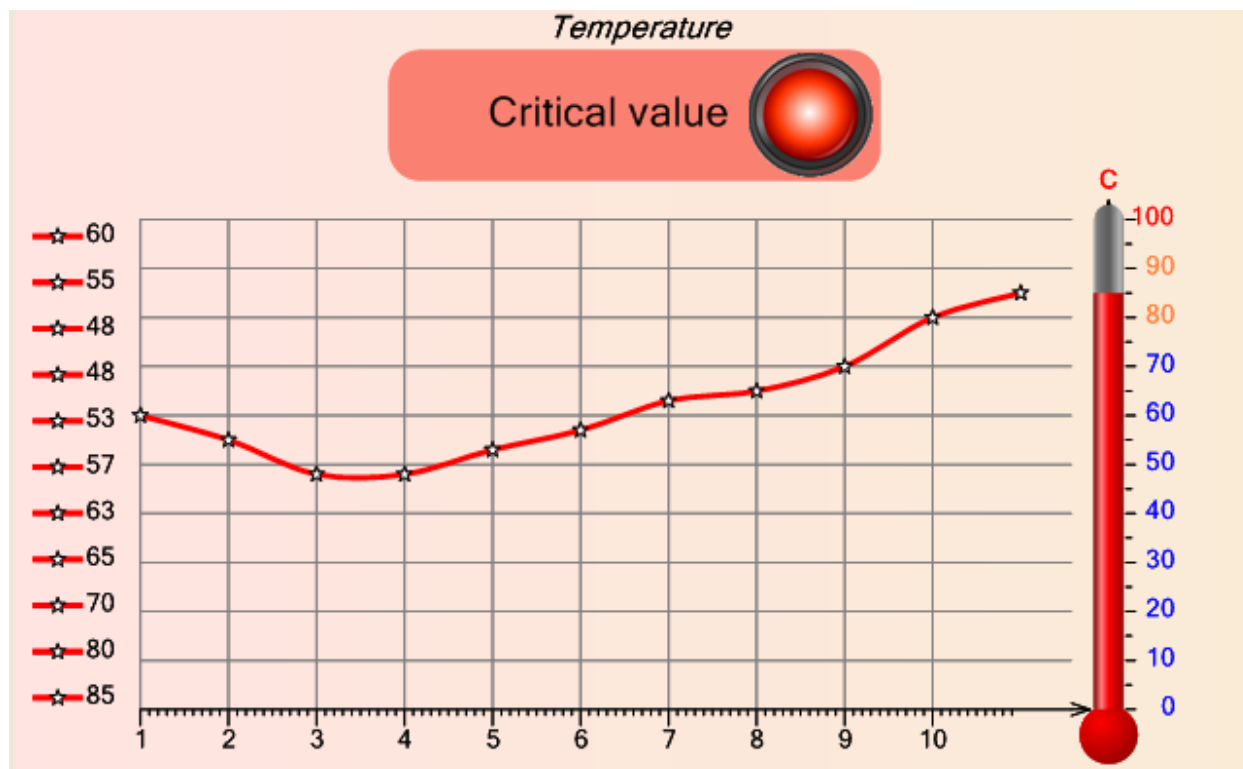
The Instrument Outline window displays current chart structure as a tree. When an element is added to the work area, it is automatically displayed in the Instrument Outline window. Selected elements are highlighted in the tree. You can change elements relative position by dragging them with a mouse in this window.





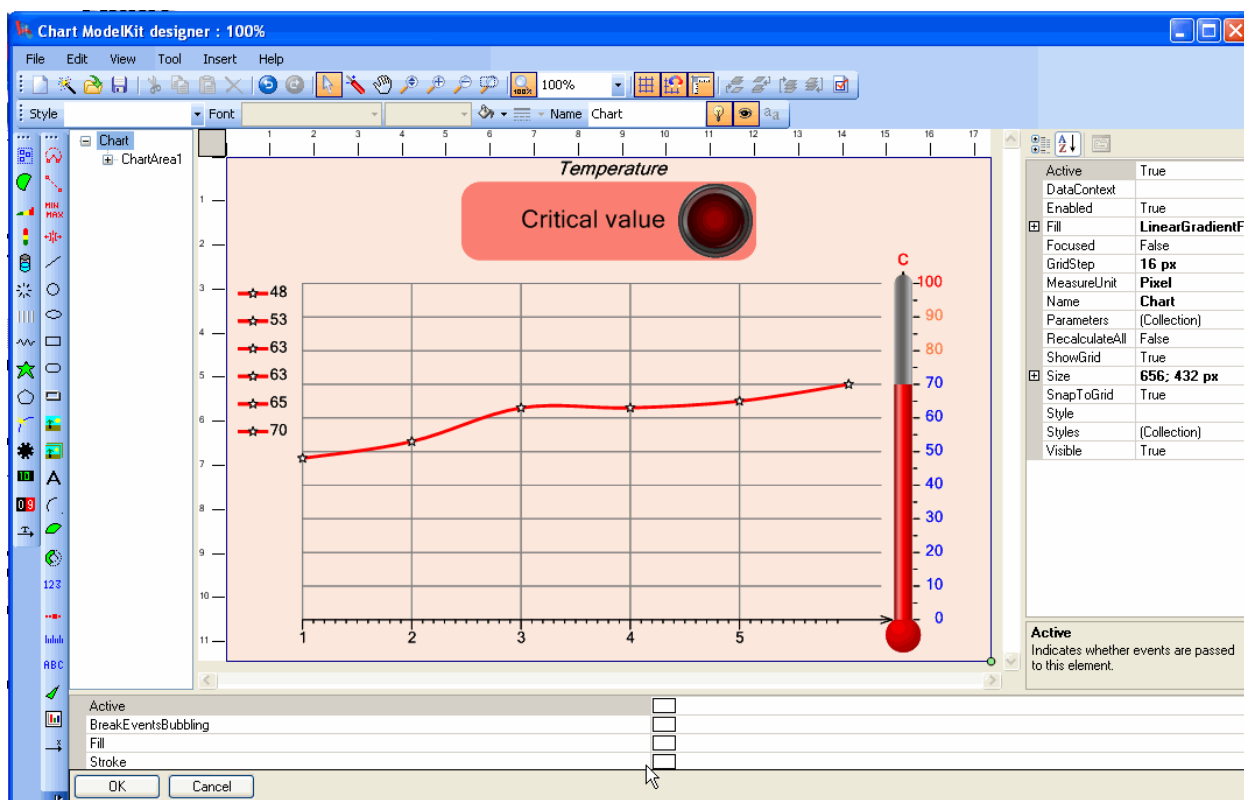
## Using DashboardViewer

The DashboardViewer control provides abilities to combine features of SharpShooter Charts charts and SharpShooter Gauges gauges. This control allows displaying of a chart, including various elements and gauges in Windows Forms.



In the picture you can see the DashboardViewer control displaying temperature value as the Thermometer gauge and the chart line displaying history of the previous temperature values. The Light indicator that lights up when the current temperature value is higher then the defined one, is located in the chart as well. One and the same axis is used for the chart and Thermometer indicator. This example can be found in the Chart Sample Center: the Using DashboardViewer node.

The DashboardViewer control provides visual designer that allows the creation, customization of charts and gauges and combination of this elements. In the Designer, all SharpShooter Gauges gauge elements and all SharpShooter Charts chart elements are available. Expressions for the element properties and the TestTool mode for testing interactive instrumentas are available as well.



The use of the DashboardViewer is analogous to the use of the ChartViewer. See "The Chart Viewer" chapter herein. The use of the designer is analogous to the use of the designer for creating charts or instruments. See "The Designer" chapter in the SharpShooter Charts section or "Use and Capabilities of the Instrument Designer" chapter in the SharpShooter Gauges section. The technique of creation instruments is similar to the SharpShooter Gauges, and the technique of creation charts is similar to the SharpShooter Charts.

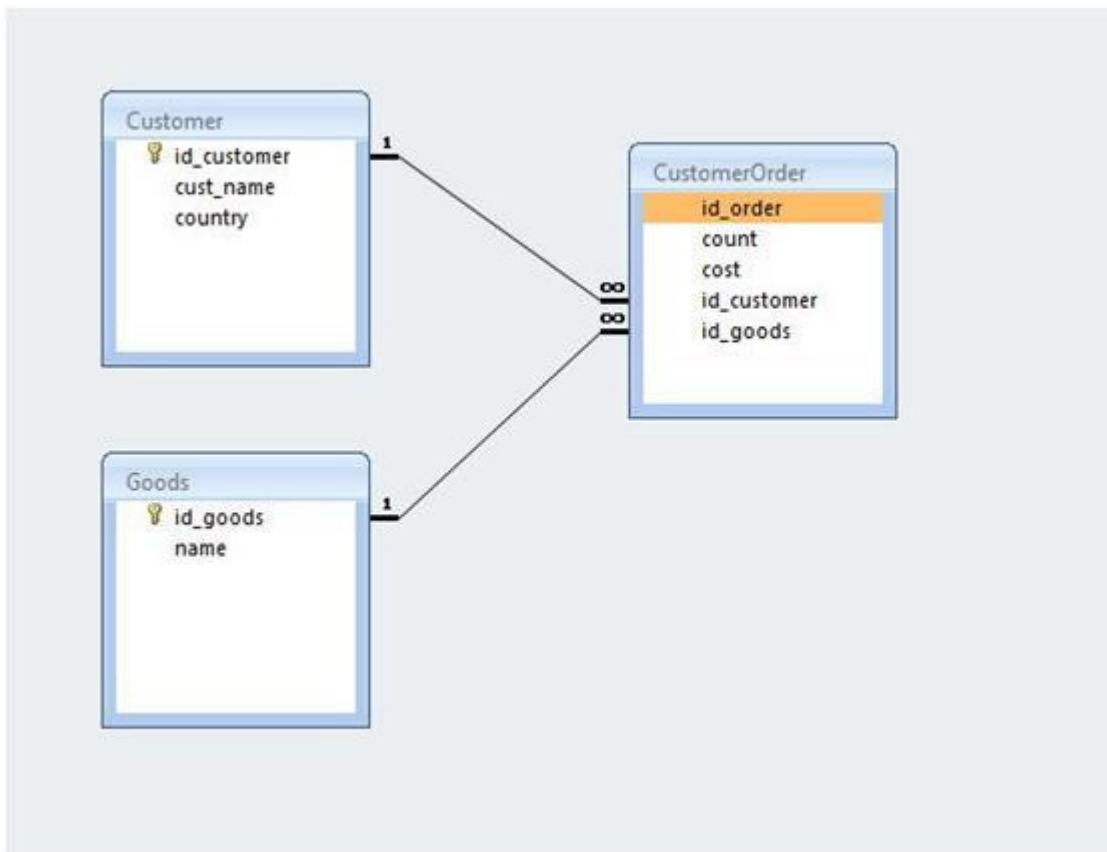
The joint use of instrument elements and chart elements doesn't impose any limitations: you can use elements in any combinations. The only condition is the Chart object that should be a root element of the composition: this doesn't prevent the Chart element from its use as an instrument as it is a descendant of the Instrument.



## Joint use of SharpShooter Charts and SharpShooter OLAP

This section contains information on joint use of SharpShooter OLAP and SharpShooter Charts. We will create a simple application that will help you understand how to use this function.

First of all we need to define data that will be used. A small database is created in Access (named it work.mdb). Database schema is displayed in the image below.



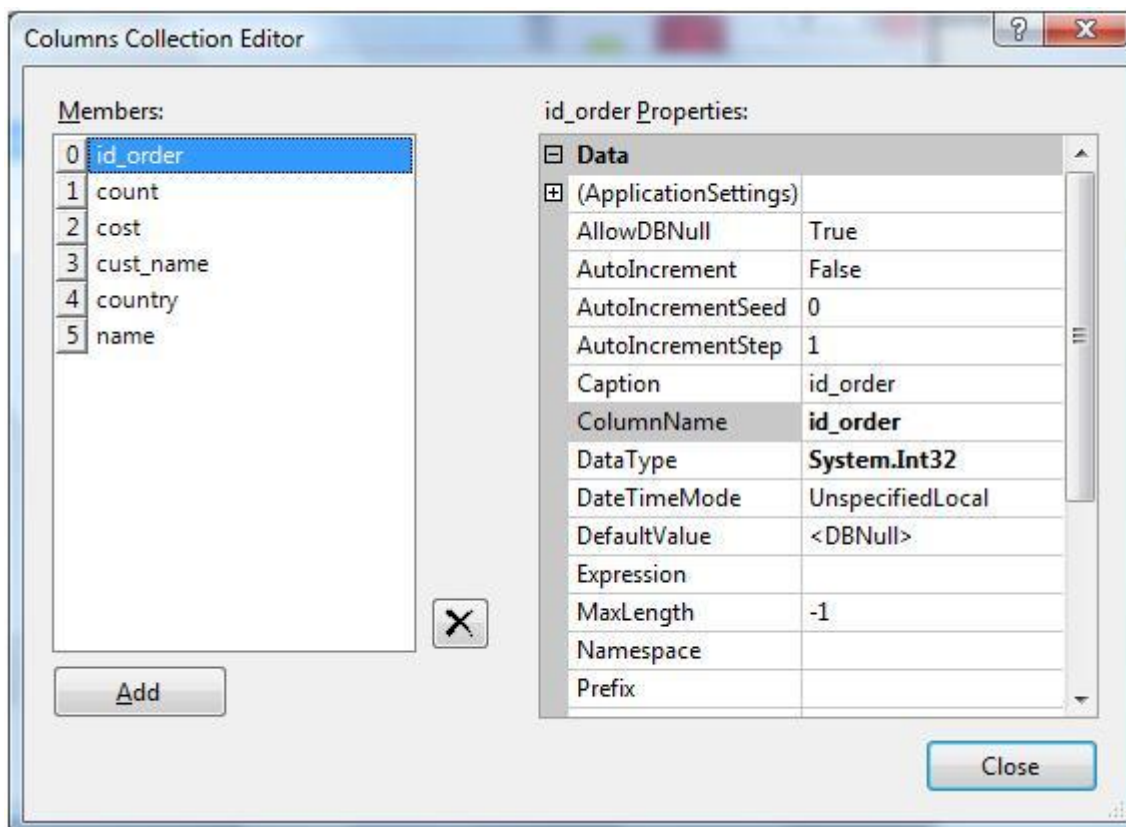
There are three tables: goods (Goods), customers (Customer), and orders (CustomerOrder). All keys are of int32 type; the rest of fields except of Count and Cost are strings.

We will create a pivot table that will display sum spent for every type of goods by each order. Then we will create a chart based on this pivot table.

	Order ID / ▼				
Goods / ▼	1	2	3	4	Total
bread	1704,221	111,051	4678,303	401,574	6895,148
cucumber	15,000	4379,192	3424,235	1199,232	9017,660
potato	318,526	487,959	1505,172	2717,760	5029,417
Total	2037,746	4978,203	9607,710	4318,566	20942,225

Let's create application.

First of all we need to think about structures that will be used as data sources. DataSet will be used for this purpose. Create a table (CustomerOrder) in the DataSet. A list of its fields is displayed in the picture below.



Cust\_name, country and name fields are strings, count and cost are of Double type, and id\_order is of int32 type.

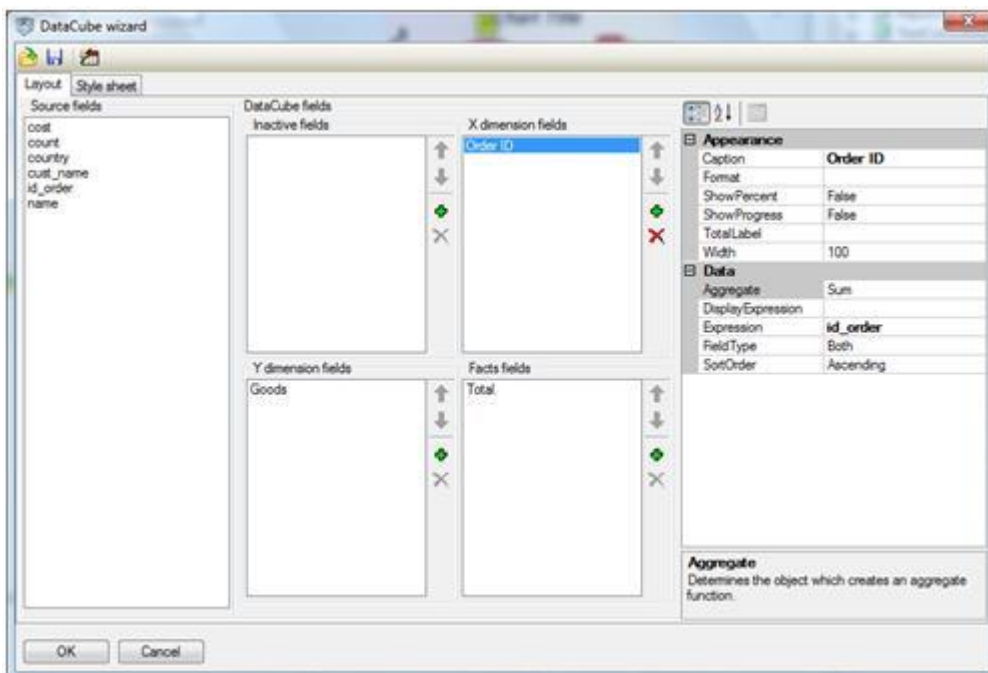
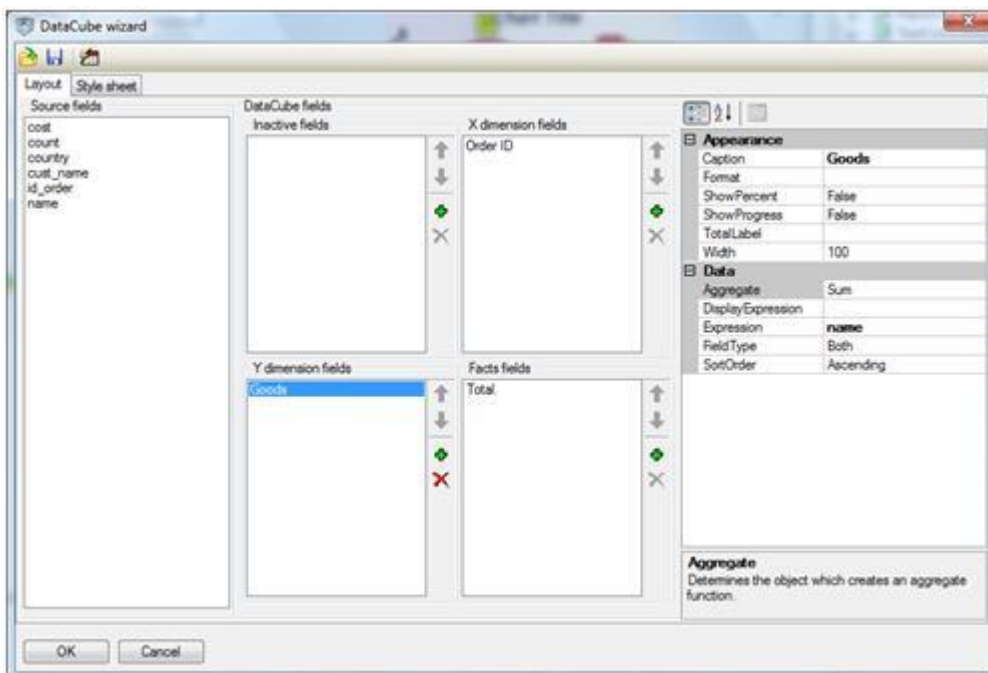
It is necessary to fill the DataSet in the OnLoad event using our database. Let's write the following code:

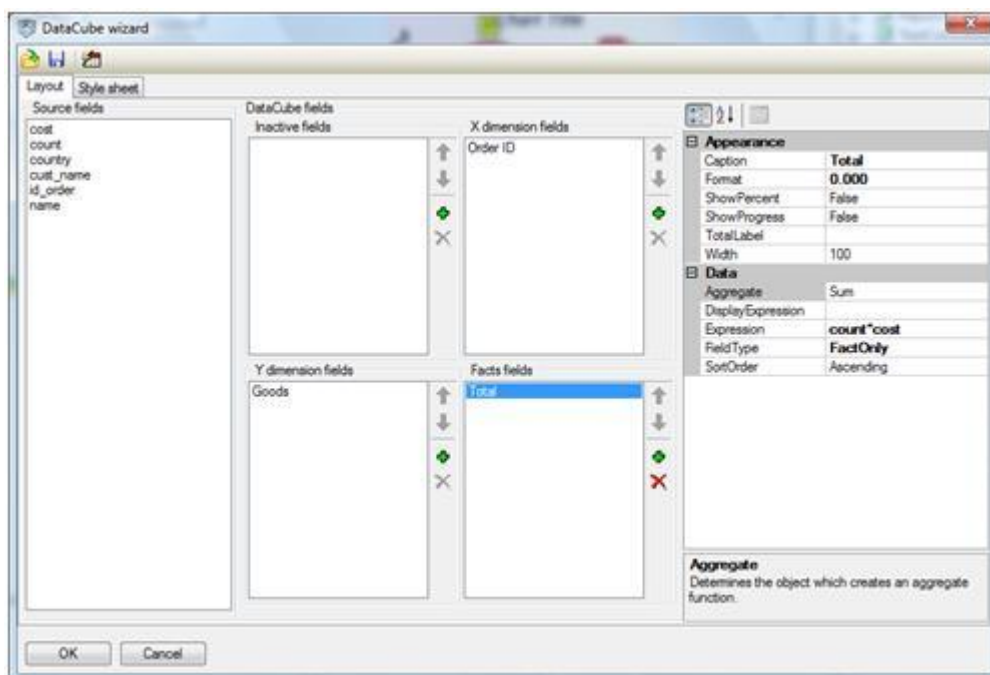
```
private void Form1_Load(object sender, EventArgs e)
{
    OleDbConnection Connection = new
    OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=\\\"..\\\"..\\\"Work.mdb\\\"");
    OleDbDataAdapter adapter2 = new OleDbDataAdapter("SELECT id_order, count,
cost, cust_name, country, name FROM (CustomerOrder INNER JOIN Customer ON
CustomerOrder.id_customer = Customer.id_customer) INNER JOIN goods on
CustomerOrder.id_goods = Goods.id_goods", Connection);
    adapter2.Fill(customerOrder);
    Connection.Close();
}
```

DataSet is ready for use. Let's add BindingSource and specify the created DataSet as a data source. Don't forget to select the CustomerOrder table as DataSet property value.

Add two components onto the form: DataCube and DataCubeGrid.

Set the created BindingSource as a datasource for DataCube (the DataSource property) and specify the following parameters for creating a pivot table:





Set this DataCube as the DataCubeGrid.Source property value.

Thus, we get pivot table. Its columns contain orders ID and rows contain description of goods.

We need to create a chart based on this table and which X axis enumerates order ID and Y axis displays sum by order. We need to do it so that filtering total data by columns and rows makes chart regenerate.

Step1: Add the DataCubeView component onto the form. Set a DataCube from which the data will be taken (the DataCube property).

Now we need to set the Argument and Value properties. There are two variants. In the first variant it is possible to set GenericCaption as argument and GenericFact as Value. GenericCaption is taken from the axis specified in the DataCubeView and GenericFact is total value by this GenericCaption. In this case GenericCaption is id\_order and GenericFact is sum by the order with id\_order ID.

Note: You can set field names as argument and value, though if you modify X-Dimension, Y-Dimension or Fact values, you will need to modify chart as well. This is not very convenient, that is why we strongly advise you to use GenericCaption and GenericFact.

Now we only need to add text to the handler of the change event of the DataCubeView (the ListChanged event). This event performs when the pivot table is recalculated including the case of filtering data. So, by writing in it text to refresh chart we will be able to regenerate chart every time data in the table is changed. Moreover, it is necessary to set method for recalculating DataCube when the application is launched. Add it to the OnLoad form handler.

Thus, we get its final variant.

```
private void Form1_Load(object sender, EventArgs e)
{
    OleDbConnection Connection = new
    OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=\\..\\..\\Work.mdb\\");
    OleDbDataAdapter adapter2 = new OleDbDataAdapter("SELECT id_order, count, cost,
    cust_name, country, name FROM (CustomerOrder INNER JOIN Customer ON
    CustomerOrder.id_customer = Customer.id_customer) INNER JOIN goods on
    CustomerOrder.id_goods = Goods.id_goods", Connection);
    adapter2.Fill(customerOrder);
}
```



```
Connection.Close();
```

```
dataCube1.Recalculate();
```

```
}
```

Here is a sample code for DataCubeView refresh handler.

```
delegate void DataChangedDelegate();
```

```
private void dataCubeView1_ListChanged(object sender, ListChangedEventArgs e)
```

```
{
```

```
    DataChangedDelegate dataChangedDelegate = chartViewer1.Chart.DataChanged;
```

```
    chartViewer1.BeginInvoke(dataChangedDelegate);
```

```
}
```



## Work in Web Forms Applications

### Using charts in Web forms applications

The WebChartViewer component is used to display a chart on the Web forms.

This component allows you to bind a chart to a data source and the display the created chart on a web page. The WebChartViewer is a DataBoundControl descendant, and this provides an ability to use any data sources available for the DataBoundControl, for example, various DataSourceControl (SqlDataSource, XmlDataSource, ObjectDataSource, etc.). The DataSourceID property is used to set a data source.

A chart is represented as an image on the final HTML page.

The WebChartViewer provides the ability to assign format of an image representing a chart (the PictureFormat property). This property can be set to one of the following values:

Jpeg - Joint Photographic Experts Group format;

Png - Portable Network Graphics format;

Gif - Graphics Interchange Format;

Bmp - Bitmap format;

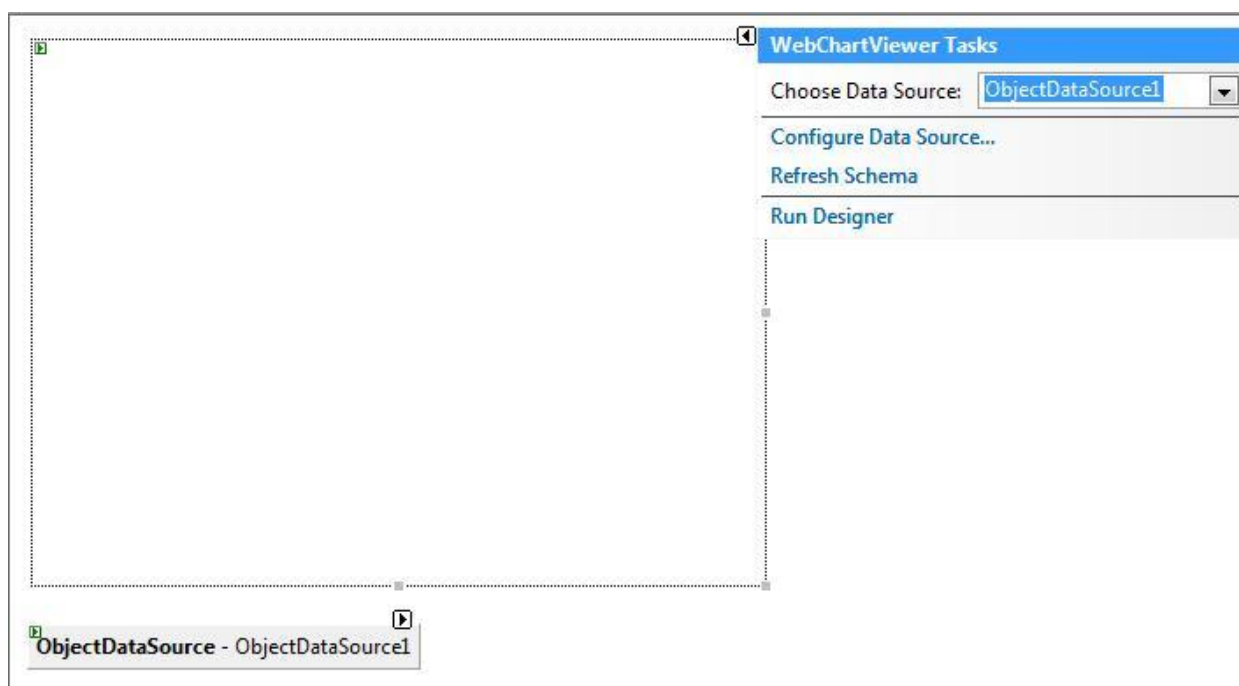
Svg - Scalable Vector Graphics format;

Flash - Flash format.

If the JPEG format is assigned, the JpegQuality property assigns quality of the Jpeg image.

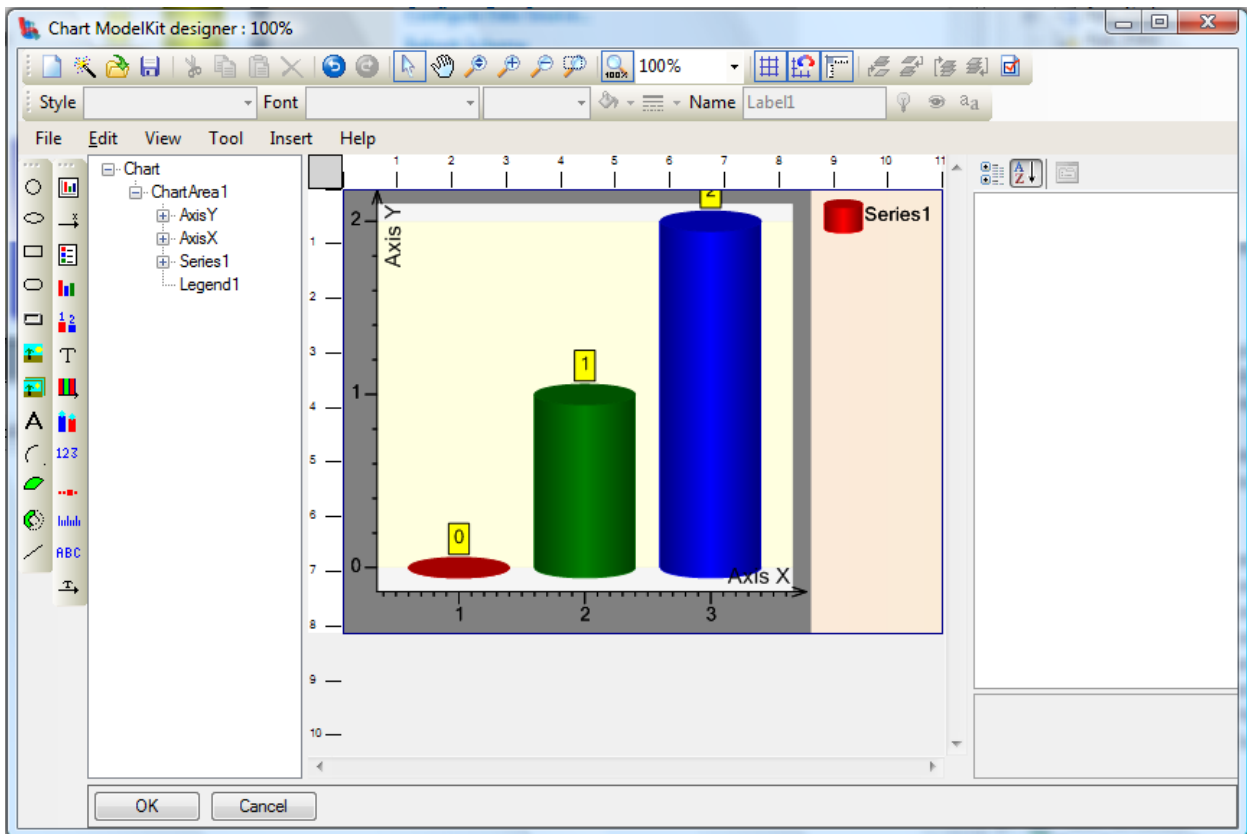
### Example of using charts

Place the WebChartViewer component located on the SharpShooter Charts tab onto ASP.NET page; assign its position and size. On the smart tags panel, select the "Choose Data Source" item and assign a data source for a chart. For example, let's assign ObjectDataSource that is a set of points assigned by the coordinates as data source.





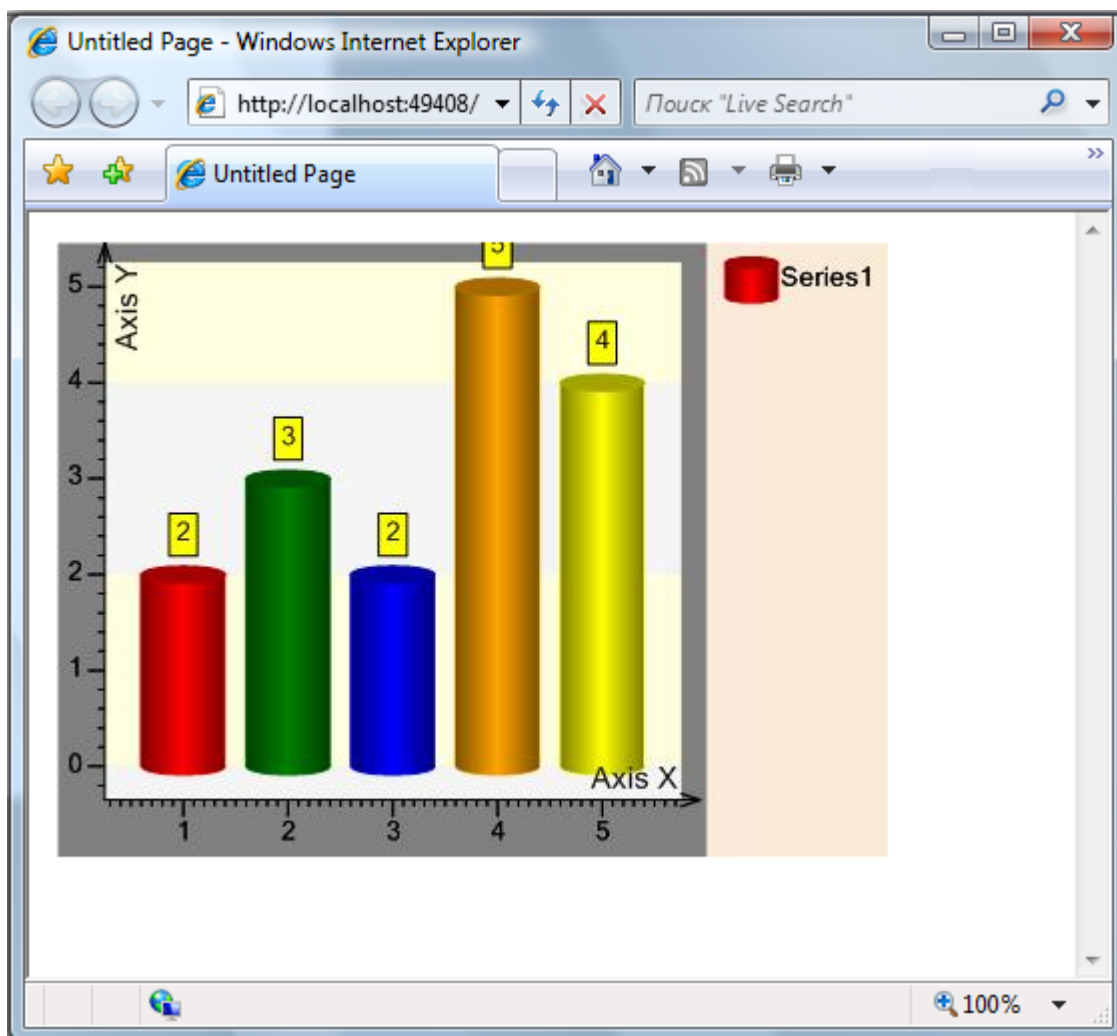
Select the "Run Designer" item on the smart tag panel; run chart designer where the displayed chart is assigned.



The designer and technique of building charts is similar to those of ChartViewer.

Assign image format in the PictureFormat property, let it be Jpeg.

Run the resulting application; a page displays chart designed by data from the data source.

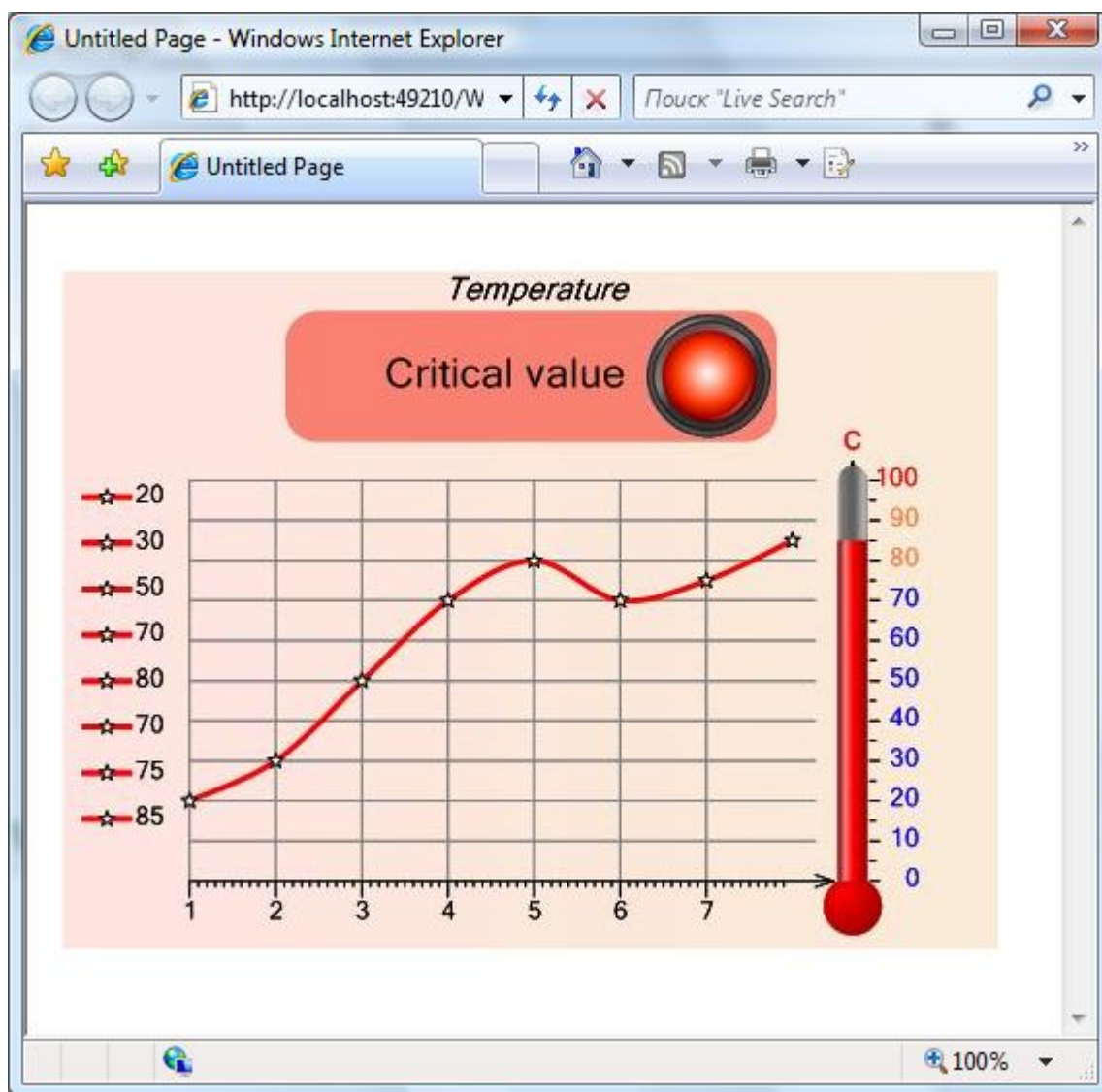


### Using dashboard in Web forms application

The WebDashboardViewer component is used to provide joint use of charts and instruments in Web forms applications. This component is similar to the DashboardViewer, but it is intended for displaying dashboards on Web forms.

The WebDashboardViewer component is a WebChartViewer descendant; that is why its use is similar.

The WebDashboardViewer component allows the simultaneous display of chart and instruments.



The WebDashboardViewer designer functionality is extended; it is capable of creating both charts and instruments simultaneously.



Chart.cmk - Chart ModelKit designer : 100%

Style Font Name AxisY

File Edit View Tool Insert Help

Chart

- Chart.Area1
  - RoundedRectangle1
  - AxisY
  - AxisX
  - Series1
    - Title1
    - Legend1
    - PictureSet1
    - Label1

Property	Value
Active	True
Arrow	PerpetuumSoft.Charts.S
AxisIndent	Auto
BreakEventsBubbling	False
Colorizer	PerpetuumSoft.Instrume
CrossValue	100%
Direction	Y
Fill	[None]
IsDiscrete	False
Margins	0; 0; 0; 0 px
MaxValue	100
MinValue	0
Name	AxisY
PerpendicularAxis	AxisX
RecalculateAll	False
Reverse	False
Smooth	True
StartFromZero	True
Stroke	SimpleStroke
Style	
Visible	True

Active  
Indicates whether events are passed to this element.

Active

BreakEventsBubbling

Fill

MaxValue

MinValue

Stroke

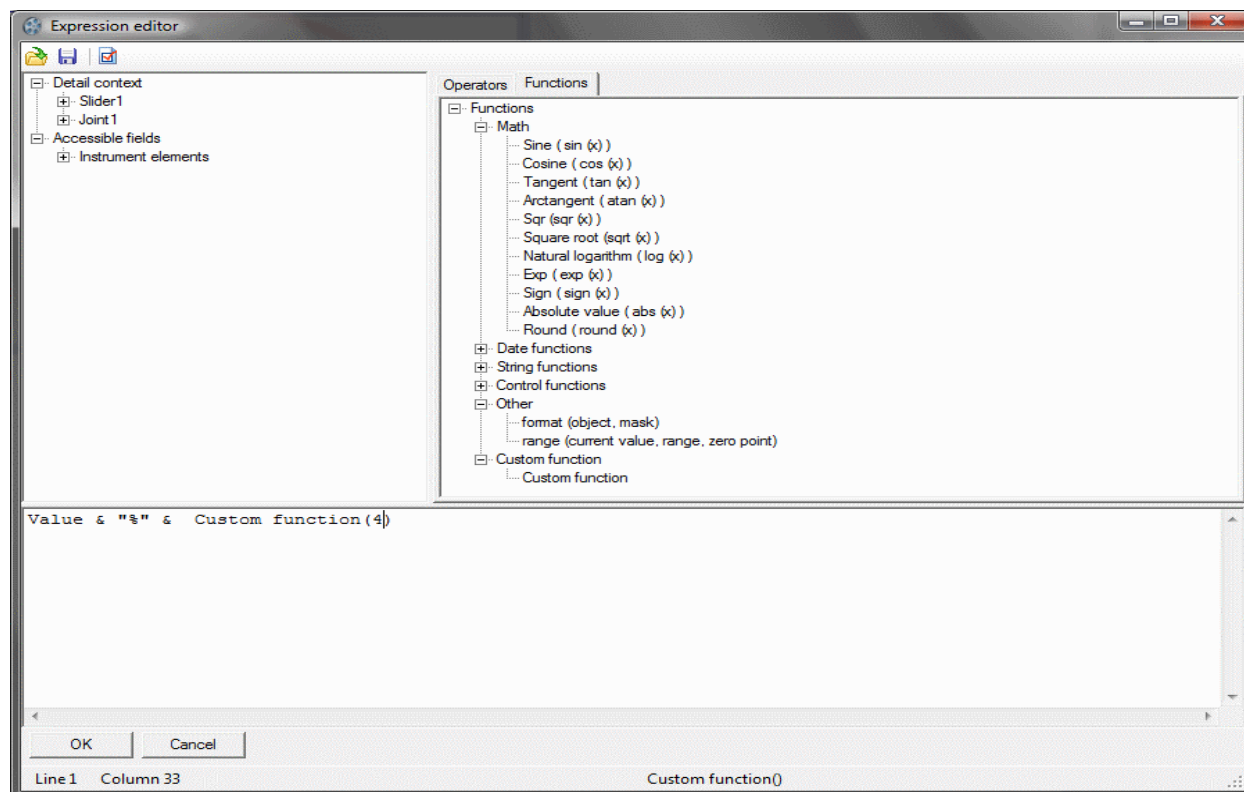
Style

Visible

OK Cancel

## Appendix 1 Working with the Expression Editor

Expression editor appearance is displayed in the image below.



Classic operators and expression language functions are located by categories in the TreeView elements on the Operators and Functions tabs. The TreeView located on the right of the window contains available data source fields subject to their nesting. Any construction described in either Tree can be moved to the text entry field by drag-and-drop. Double click on the tree element will lead to pasting the construction into the current position of the entry field.

After the entry is complete it is required to click the "OK" button to confirm modifications or the "Cancel" button to cancel modifications. If the "OK" button is pressed, syntax checking is executed before the window is closed.

Also you can create your own functions which will be displayed and dragged as the rest ones. You can calculate values that you need by specified parameters. It is necessary to create class that realizes `PerpetuumSoft.Framework.Expressions.ICustomFunction` interface. After that you should register

`PerpetuumSoft.Framework.Expressions.BuiltInFunctions` in the static class in the following way:


```
PerpetuumSoft.Framework.Expressions.BuiltInFunctions.RegisterFunction("Custom function", new CustomFunction());
```

The expression editor has a button panel.

The "Save" button (  ) allows saving expression text to a file.

The "Open" button (  ) allows reading expression text from a file.



The “Check expression” button (  ) allows executing syntax checking. In case a syntax error is found a corresponding warning specifying the initial position of the erroneous construction is displayed.